ACORN: COMPLIANT AND EFFICIENT LLM AGENTS VIA CONTRACT-BASED SYMBOLIC CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents augmented by tools are increasingly deployed to execute standard operating procedures (SOPs), acting on external systems through tool calls. However, prompt-based guidance leaves compliance to the model, while blocking guardrails may lead to repeatedly rejected actions without advancing the task. In addition, when the SOP rules are sufficient to already determine the next call, querying the model only adds error and cost. We present ACORN (Agentic COntract-based Runtime coNtrol), an agent framework that leverages contract-based symbolic control to address these challenges. The required behaviors are formalized as assume-guarantee (A/G) contracts in linear temporal logic on finite traces (LTL_f) over the agent's tool-call trace, i.e., the sequence of tool calls and returns. The contracts are compiled to deterministic finite automata (DFAs). Given the fixed tool set of an agent session, the framework uses the DFAs before each tool call to select the subset of tools that the contracts currently permit and exposes it to the model. When the subset leaves only one possible call, the call is executed without consulting the model. Any tool call that would violate a contract or leave some contract impossible to satisfy is blocked. Each selected subset of tools is cached and reused in later sessions, whenever they reach the same DFA states and recorded values. On ten SOP-Bench domains, ACORN commits no procedural violation, improves macro task success by 23.1 percentage points, and reduces estimated cost by 62%. On two τ^2-bench domains, it improves pass^4 by 15.0 points on average.

1 INTRODUCTION

Tool-augmented large language model (LLM) agents are increasingly deployed to execute predefined procedures, such as standard operating procedures (SOPs) in logistics and compliance, customer-service policy documents, and clinical intake protocols (Li et al., 2025; Nandi et al., 2025; Barres et al., 2025). In these settings the agent's sequence of tool calls must itself comply with the procedure, which may prescribe verification before approval, mandated follow-ups, and prohibitions that must hold regardless of the context. Moreover, compliance should be decided at the *action boundary*, the instant before a tool call takes effect, when its consequences become irreversible.

Most existing approaches act either before a tool call is proposed or after it has taken effect. Alignment training acts before deployment (Casper et al., 2023), and trained models may still game a grader rather than solve the task (Amodei et al., 2016; Skalse et al., 2022; Baker et al., 2025) or take actions they identify as prohibited when those actions serve an assigned goal (Lynch et al., 2025). It is also fixed before the procedure is known, so it cannot settle a tool call that one SOP mandates and another forbids. Post-hoc auditors grade a trajectory after its side effects have occurred (Zheng et al., 2023; Sharma et al., 2026). Prompt-level guidance restates the procedure to a model that remains free to ignore it. Runtime guardrails do act at the action boundary, but they only block a proposed call (Wang et al., 2026; Inan et al., 2023). The model still decides what to propose, and the guardrail can only block the call and ask again. Tool filters and workflow engines narrow the tools the model sees or execute fixed steps without it (Ye et al., 2025; Qiu et al., 2025; Babu & Iyer, 2026), but they rely on a hand-written graph or a relevance heuristic, and none derives these decisions from the procedure's rules with a compliance guarantee.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

The failures we observe on SOP-Bench (Nandi et al., 2025) show why blocking a proposed call is not enough. In several domains, the model (i) orders steps incorrectly, (ii) executes the right steps but corrupts their outputs when transcribing them into the final report, (iii) takes actions the SOP forbids, and (iv) loses track of the procedure in a tool set with many irrelevant tools. A block-and-reprompt guardrail addresses (iii) at best, and it can even lower task success. In each case the procedure fixes the correct call or excludes the wrong one, yet the model still chooses, leaving room for error.

We present ACORN (Agentic COntract-based Runtime coNtrol), a neuro-symbolic agent control framework that compiles a procedure into runtime control (Figure 1). The rules of the procedure, authored from its documents or mined from traces, form a *contract library* of assume-guarantee (A/G) contracts in temporal logic over the agent's tool-call trace (Elkoussy & Perez, 2026; Xiao & Nuzzo, 2026), and each contract is compiled into an automaton. Given the fixed tool set of the agent, the states of these automata drive three stages before every tool call. *Masking* exposes to the model only the tools whose call the contracts currently admit. *Contract fast-forward* executes the call itself, with no model call, when exactly one admissible tool remains and earlier calls or returns determine its arguments. *Verification* checks every call, proposed by the model or issued by ACORN, against the contracts before it executes. Our contributions are as follows:

- We present ACORN, an agent control framework that enforces contracts at runtime, with a new completion verdict that evaluates whether every contract can still be satisfied after a proposed call.
- We introduce dynamic tool masking, which proactively restricts the tools exposed to the model to those the contracts admit and caches these decisions across sessions, and contract fast-forward, which executes the next call without consulting the model when the contracts determine it.
- Experiments on ten SOP-Bench domains show that ACORN raises macro task success by 23.1 percentage points (pp) at 62% lower estimated cost without a single procedural violation, and on two τ^2-bench domains it raises pass^4 by 15.0 pp on average.

2 PRELIMINARIES

Agent Frameworks. An agent pairs an LLM backbone with a tool set $\mathcal{T}$, issuing tool calls and folding their outputs into a reasoning loop until task completion (Yao et al., 2023). The *tool-call interface* is the set of tools the framework exposes to the agent, together with their call and return conventions, and the sequence of events the agent emits at this interface is its *tool-call trace*.

Definition 1 (Agent session) *Let Σ denote the finite set of* events *at the tool-call interface. Each tool call produces two events: a* call *event $a = (T, args)$ when the agent issues the call to a tool $T \in \mathcal{T}$, and a* return *event $a' = (T, result)$ when the executed tool returns. An agent's execution is a transition system (Baier & Katoen, 2008) (S, s_0, Σ, R). A state $s := (\tau, \kappa, \rho) \in S$ comprises the trace $\tau \in \Sigma^*$ of events issued so far, a finite map κ from context keys to values recorded from earlier events, and a vector ρ of numeric session counters, with $s_0 := (\varepsilon, \kappa_0, \rho_0)$. Each event a induces a transition $(s, a, s') \in R$ that appends a to τ and updates κ and ρ. An agent* session *is a finite path $s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_k} s_k$, with trace $\tau = a_1 \cdots a_k$.*

A context key records a fact carried across events, such as the caller's identity or a document's source, and a counter tracks a cumulative quantity, such as a per-tool call count. Let $\Delta(X)$ denote the probability simplex over a finite set X, and let $\Sigma_c \subseteq \Sigma$ be the call events, so that $\Sigma \setminus \Sigma_c$ are the return events. For $\mathcal{T}' \subseteq \mathcal{T}$, we write $\Sigma_c(\mathcal{T}')$ for the calls of its tools. The *LLM-based policy* is a map $\pi_{\text{LLM}} : S \times 2^{\mathcal{T}} \to \Delta(\Sigma_c)$, whose second argument is the set of tools offered to the model.

Assume-Guarantee (A/G) Contract. Let M denote a component, i.e., an element of a system, characterized by a set of variables V_c and a set of behaviors $[\![M]\!]$ over V_c. A contract C formally captures a set of specifications for M using a triple $C = (V_c, A, G)$ (Benveniste et al., 2018), where A and G are sets of behaviors over V_c. A is the assumptions on the environment of M while G is the guarantees provided by M, given that the assumptions are satisfied. We say that M is a valid implementation of C, i.e., $M \models C$, if all the behaviors of M are included in the guarantees given the assumptions of C, i.e., $[\![M]\!] \subseteq G \cup \overline{A}$. We say that component E_c is a valid environment of C if all the behaviors of E_c are contained in the assumptions of C.

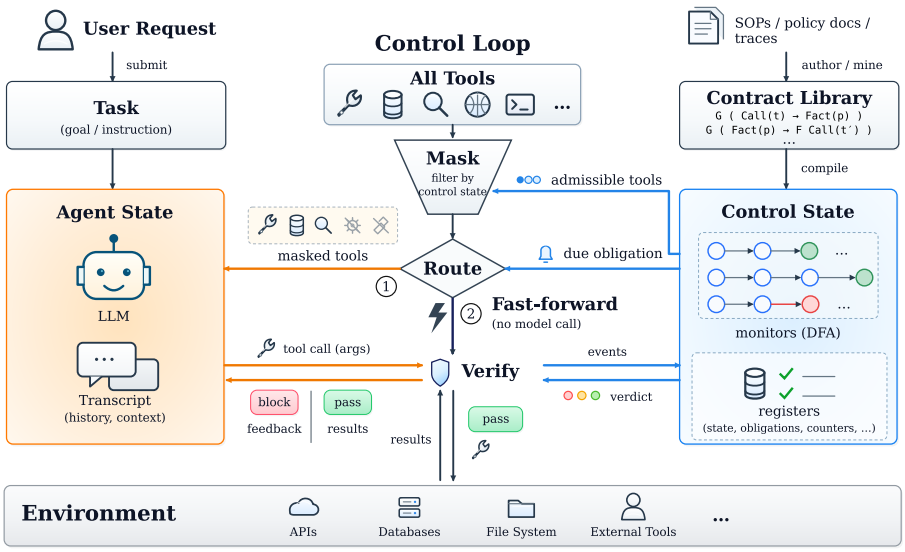


Figure 1: Overview of the ACORN framework.

Finite-Trace LTL (LTL_f). A session of Def. 1 is finite, so its properties are written in linear temporal logic on finite traces (LTL_f) (De Giacomo & Vardi, 2013). LTL_f formulae over a finite set V of atomic propositions (AP) follow the grammar $\varphi ::= P \mid \neg\varphi \mid \varphi \wedge \varphi' \mid X\,\varphi \mid F\,\varphi \mid G\,\varphi \mid \varphi\,U\,\varphi'$, where $P \in V$, with temporal operators X (next), F (eventually), G (always), and U (until). Each formula φ compiles to a deterministic finite automaton (DFA) $D_\varphi = (Z, z^0, 2^V, \theta, Z^{\mathrm{acc}})$ with a finite state set Z, an initial state $z^0 \in Z$, the valuations 2^V as alphabet, a transition function $\theta : Z \times 2^V \to Z$, and accepting states $Z^{\mathrm{acc}} \subseteq Z$. Its extension to finite sequences of valuations is $\theta^*(z, \nu_1 \cdots \nu_k) := \theta(\cdots\theta(\theta(z, \nu_1), \nu_2)\cdots, \nu_k)$ for $z \in Z$ and $\nu_1, \ldots, \nu_k \in 2^V$, with $\theta^*(z) := z$ for $k = 0$, and D_φ accepts a sequence $\nu_1 \cdots \nu_k$ exactly when $\theta^*(z^0, \nu_1 \cdots \nu_k) \in Z^{\mathrm{acc}}$, that is, when the sequence satisfies φ. This is the canonical monitor construction of runtime verification (Bauer et al., 2011). Atomic propositions that compare a quantity of the session state against a constant bound make a formula one of arithmetic LTL_f (arithmetic LTL_f (ALTL_f)) (Felli et al., 2023), an instance of LTL_f modulo theories (Geatti et al., 2022). Each such comparison takes a truth value at every event from the session state, so the DFA treats it as an ordinary proposition. The arithmetic matters only when reasoning about events not yet observed.

ACORN instantiates the atomic propositions of V as *interaction predicates* over the agent's tool-call trace. V is partitioned as $V_{\mathrm{ag}} \cup V_{\mathrm{env}}$, mirroring the distinction between controlled and uncontrolled variables in contract-based design (Benveniste et al., 2008). The assumption is an ALTL_f formula over V_{env} and the guarantee an ALTL_f formula over V. Each predicate takes a value at every event of the session, so a trace τ induces a sequence of valuations of V on which formulas are evaluated.

Definition 2 (Runtime contract valuation) *Against a concrete trace τ, the valuation of a contract $C = (V, A, G)$ takes one of three values, named* IDLE *(1),* ACTIVE *(e), and* FAIL *(0), ordered as $0 \le e \le 1$, following the multi-valued verdicts of runtime verification (Bauer et al., 2006):*

$$[\![(A, G)]\!](\tau) = \begin{cases} 1, & \tau \models \overline{A}, \\ e, & \tau \models A \wedge G, \\ 0, & \tau \models A \wedge \overline{G}. \end{cases} \tag{1}$$

A procedure is rarely one rule. For a finite set of contracts $\mathcal{C} = \{C_1, \ldots, C_n\}$, the verdict on τ is the pair $v_{\mathcal{C}}(\tau) = \big(v_{\mathrm{env}}(\tau),\ v_{\mathrm{ag}}(\tau)\big)$ of the join and the meet of the per-contract valuations,

$$v_{\mathrm{env}}(\tau) = \bigvee_i [\![(A_i, G_i)]\!](\tau), \qquad v_{\mathrm{ag}}(\tau) = \bigwedge_i [\![(A_i, G_i)]\!](\tau). \tag{2}$$

The environment has broken its side when $v_{\mathrm{env}}(\tau) = 1$, since some assumption has been falsified, and the agent has violated a contract when $v_{\mathrm{ag}}(\tau) = 0$.

3 THE ACORN FRAMEWORK

ACORN runs the loop of Figure 1 once per step. *Mask* derives the tools the model may see (§3.3), *route* decides whether to consult the model or to fast-forward (§3.4), and *verify* checks every call before and after it runs (§3.2). The three stages realize the three maps of the problem stated in §3.1, and the end of §3.4 composes them into one decision policy.

3.1 PROBLEM FORMULATION

A contract library $\mathcal{C} = \{(A_i, G_i)\}_{i=1}^n$ is a set of agent contracts in ALTL$_f$ obtained from a procedural document. It is written against the tool interface and the procedure, so one library serves any agent that exposes the same tools. We assume that the library is conflict-free, and leave the correctness of its authoring out of scope. Each A_i and G_i compiles to a DFA over the valuations of the predicates V (De Giacomo & Vardi, 2013).

Definition 3 (Control state) *Let $\mathcal{C} = \{(A_i, G_i)\}_{i=1}^n$ be a contract library over predicates V, and let D_{A_i} and D_{G_i} be the DFAs compiled from A_i and G_i. The* control state *of $s = (\tau, \kappa, \rho) \in S$ is*

$$\hat{s}(s) := \big(z_1^A, \ldots, z_n^A,\ z_1^G, \ldots, z_n^G,\ \kappa_V,\ \rho_V\big), \tag{3}$$

where $z_i^A := \theta_{A_i}^*(z_{A_i}^0, \nu_1 \cdots \nu_k)$ and $z_i^G := \theta_{G_i}^*(z_{G_i}^0, \nu_1 \cdots \nu_k)$ are the states that D_{A_i} and D_{G_i} reach on the valuations $\nu_1 \cdots \nu_k$ that τ induces, with z^0 and θ the initial state and transition function of each DFA, and κ_V and ρ_V are the projections of κ and ρ onto the context keys and counters that occur in the predicates of V.

The control state for an agent state replaces the trace τ with finitely many DFA states and keeps only the projections of κ and ρ onto what the predicates of V refer to, so sessions with different histories can share a control state. The verdict of Eq. (2) evaluates the trace observed so far. Writing $\tau\tau'$ for τ followed by a finite, possibly empty, sequence $\tau' \in \Sigma^*$ of events still to come, a *continuation* of τ, a trace with $v_{\mathcal{C}}(\tau) = (e, e)$ may have no τ' with $v_{\mathcal{C}}(\tau\tau') = (e, e)$, e.g., when two contracts each require a different call at the next event, a *conflict* that no per-contract verdict detects (Maggi et al., 2012). The *completion verdict* $u_{\mathcal{C}}(\tau)$ is $\top$ if

$$v_{\mathcal{C}}(\tau\tau') = (e, e) \quad \text{for some } \tau' \in \Sigma^*, \tag{4}$$

and $u_{\mathcal{C}}(\tau) = \bot$ otherwise, where $\tau\tau'$ is evaluated as a complete trace, written $u_{\mathcal{D}}$ when evaluated on a set $\mathcal{D}$ of contracts. The empty τ' is a witness exactly when the session may stop at τ. A trace with $v_{\mathcal{C}}(\tau) = (e, e)$ and $u_{\mathcal{C}}(\tau) = \bot$ is a *dead end*, which no continuation completes.

A call $a \in \Sigma_c$ is *legal* after τ if $v_{\mathcal{C}}(\tau\, a) = (e, e)$ and $u_{\mathcal{C}}(\tau\, a) = \top$. The tools with a legal call are

$$\mathcal{T}_{\text{legal}}(\tau) \;=\; \big\{\, T \in \mathcal{T} \;:\; v_{\mathcal{C}}(\tau\, a) = (e, e) \;\wedge\; u_{\mathcal{C}}(\tau\, a) = \top \;\text{ for some } a \in \Sigma_c(\{T\}) \,\big\}. \tag{5}$$

A *controller* for $\mathcal{C}$ consists of three maps on traces. The *admissible map* $\alpha : \Sigma^* \to 2^{\mathcal{T}}$ gives the admissible tools, which are shown to the model. The *delegation map* $\delta : \Sigma^* \to \Sigma_c \cup \{\bot\}$ gives the call executed without consulting the model, and $\bot$ when the model decides. The *enforcement map* $\sigma : \Sigma^* \to \{\mathsf{pass}, \mathsf{block}\}$ decides whether each event passes before it enters the trace.

Problem 1 (Contract-based agent control) *Given $\mathcal{T}$, π_{LLM}, and a contract library $\mathcal{C}$, construct a controller (α, δ, σ) such that, after every prefix τ of a session, (a) $\sigma(\tau\, a) = \mathsf{pass}$ if and only if a is legal after τ, for every call a; (b) $\alpha(\tau) = \mathcal{T}_{\text{legal}}(\tau)$; and (c) $\delta(\tau) = a$ if a is the only legal call after τ, and $\delta(\tau) = \bot$ otherwise.*

Intuitively, (a) blocks violations after the model proposes a call, (b) restricts the tools it can use beforehand, and (c) takes control only when $\mathcal{C}$ leaves a single legal call.

3.2 CONTRACT-BASED VERIFICATION

Verification computes both verdicts from the control state. For an event a, every DFA advances one transition on the valuation of a, which for a call includes its arguments, and $v_{\mathcal{C}}(\tau\, a)$ is evaluated from the states reached. For a formula φ and a trace τ, the *residual* $\text{res}(\varphi, \tau)$ is a formula, unique up to equivalence, such that $\tau' \models \text{res}(\varphi, \tau)$ if and only if $\tau\tau' \models \varphi$ for every $\tau' \in \Sigma^*$ (Roşu &

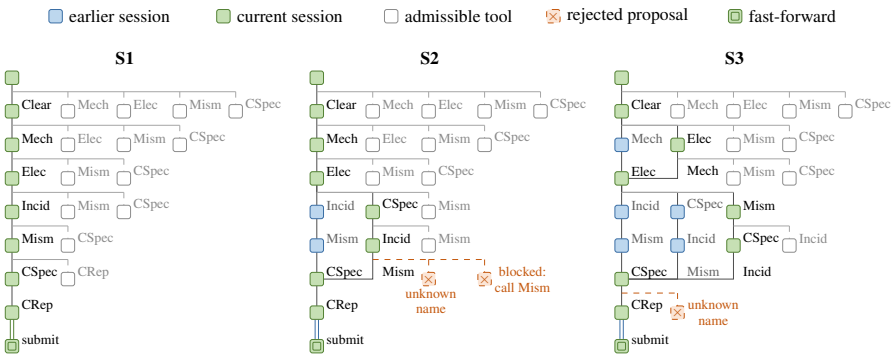


Figure 2: Three sessions on the residual policy graph, merging where they reach the same state. Blue: earlier sessions. Green: current session and its admissible tools. Orange: blocked proposal.

Havelund, 2005). For $\varphi = F\,p$ with $p \in V$, it stays φ until p holds and becomes $\top$ afterwards. The completion verdict asks whether there exists a single continuation $\tau' \in \Sigma^*$ that satisfies the residuals of all contracts,

$$u_{\mathcal{C}}(\tau\,a) = \top \iff \mathrm{SAT}\Big(\bigwedge_{i=1}^{n} \big(\mathrm{res}(\varphi_{A_i},\,\tau\,a) \wedge \mathrm{res}(\varphi_{G_i},\,\tau\,a)\big)\Big), \tag{6}$$

where φ_{A_i} and φ_{G_i} are the formulas that specify A_i and G_i, each residual is represented by the corresponding DFA state in the control state, and SAT asks for a satisfying finite trace. The enforcement map passes a call, $\sigma(\tau\,a) = \mathsf{pass}$, when both verdicts hold, $v_{\mathcal{C}}(\tau\,a) = (e, e)$ and $u_{\mathcal{C}}(\tau\,a) = \top$, and blocks it otherwise, so a call passes exactly when it is legal, which is requirement (a) of Problem 1. Calls proposed in one step are verified in order, each against the trace the earlier ones extended. An event that σ blocks never enters τ, so both verdicts hold on every prefix.

Every event passes through σ. A call event has not yet executed, so ACORN blocks it and returns to the model the contracts it would violate and, when a contract only lacks a prerequisite, the tools whose return events would supply it. A call to a tool outside $\mathcal{T}_{\mathrm{adm}}(\tau)$, including a tool that does not exist, is blocked with $\mathcal{T}_{\mathrm{adm}}(\tau)$ as the reason, so the model can choose again among the admissible tools. A return event carries a result that the environment has already produced, so ACORN instead suppresses it: the model does not receive the result and the event updates no state. Blocking a call event keeps the agent a valid implementation of every contract, and suppressing a return event keeps its environment a valid environment (Aceto et al., 2021).

3.3 Dynamic Tool Masking

Blocking a call event costs a model call, and the model may propose it again. The tool mask prevents such proposals by acting as a preemptive shield (Alshiekh et al., 2018) of the model's tool list, judging which tool call events lead to a contract violation. We first filter the predicates that are determined by the called tool alone,

$$V^{\mathrm{tool}} := \{\, p \in V \;:\; \nu_{s,a}(p) = \nu_{s,b}(p) \;\text{ for all } s \in S,\; T \in \mathcal{T},\; a, b \in \Sigma_c(\{T\}) \,\}, \tag{7}$$

where $\nu_{s,a}$ is the valuation of V at event a in session state s. Let $\mathcal{C}^{\mathrm{tool}} \subseteq \mathcal{C}$ be the contracts whose assumption and guarantee include predicates of V^{tool}. The *admissible tool set* is defined as follows,

$$\mathcal{T}_{\mathrm{adm}}(\tau) \;=\; \{\, T \in \mathcal{T} \;:\; v_{\mathcal{C}^{\mathrm{tool}}}(\tau\,a) = (e, e) \;\wedge\; u_{\mathcal{C}^{\mathrm{tool}}}(\tau\,a) = \top \;\text{ for } a \in \Sigma_c(\{T\}) \,\}, \tag{8}$$

and the mask is the admissible map $\alpha(\tau) := \mathcal{T}_{\mathrm{adm}}(\tau)$. The mask computes both conditions as verification does (§3.2), restricted to $\mathcal{C}^{\mathrm{tool}}$ and for a hypothetical call of each tool, without advancing the control state. We omit other contracts of $\mathcal{C}$, which can only be evaluated once the model proposes a call with its arguments. Hence $\mathcal{T}_{\mathrm{legal}}(\tau) \subseteq \mathcal{T}_{\mathrm{adm}}(\tau)$, which meets requirement (b) in one direction only, since a call of a tool in $\mathcal{T}_{\mathrm{adm}}(\tau)$ can still violate a contract that depends on its arguments. The verification step then checks the arguments of the call the model proposes (§3.2). The model receives $\mathcal{T}_{\mathrm{adm}}(\tau)$ as its tool schemas. Session S1 of Figure 2 shows it changing at each step.

Residual Policy Graph (RPG). Whether a tool belongs to $\mathcal{T}_{\mathrm{adm}}(\tau)$ depends on the session only through the tool and the control state. The control states reachable from $\hat{s}(s_0)$ therefore form a graph,

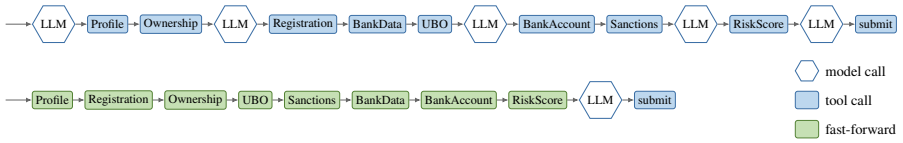


Figure 3: One session under passive (top) and under ACORN (bottom). Each hexagon is one call to the model. Green: fast-forward, no model call.

with an edge for each event that takes one to another, and each node carries the admissible set of Eq. (8). As shown in Figure 2, sessions merge when they reach the same control state. The *residual policy cache* maps each pair $(\hat{s}, T)$ of a control state and a tool to whether T is admissible at $\hat{s}$, so a later visit to that node reuses the admissible set instead of recomputing it, within a session and across sessions of the same library.

3.4 Contract-Based Fast-Forward

When only one legal call exists at a step, ACORN fast-forwards and executes the call event directly without consulting the model. For the legal call event $(T, args)$, the admissible tool set $\mathcal{T}_{\mathrm{adm}}(\tau)$ supplies the tool T when $|\mathcal{T}_{\mathrm{adm}}(\tau)| = 1$. If T takes no arguments, it is executed directly. Otherwise its arguments must be determined by the session state, which the argument map below makes precise.

Each tool T carries an *argument map* β_T, a deterministic map from the recorded facts κ and counters ρ to an argument vector for T, or to $\perp$ where these do not determine one, for instance by copying a value that an earlier call or return event recorded. ACORN executes $(T, \beta_T(\kappa, \rho))$ with no model call whenever $\mathcal{T}_{\mathrm{adm}}(\tau) = \{T\}$ and $\beta_T(\kappa, \rho) \neq \perp$, so its delegation map is

$$\delta(\tau) = \begin{cases} (T, \beta_T(\kappa, \rho)), & \mathcal{T}_{\mathrm{adm}}(\tau) = \{T\} \ \wedge \ \beta_T(\kappa, \rho) \neq \perp, \\ \perp, & \text{otherwise}, \end{cases} \tag{9}$$

where κ and ρ belong to the session state whose trace is τ. $\beta_T(\kappa, \rho)$ can be considered as a prediction of the arguments that the model would have chosen for tool T given the current session state, so it needs to be verified before execution (§3.2). Fast-forward meets requirement (c) only at the level of tools: β_T may fix an argument that the contracts leave open, and when a single legal call exists but β_T returns $\perp$, ACORN consults the model. Figure 3 shows the effect on one session, where eight of nine steps run without the model.

A contract can also require a call of a tool T that has not happened yet, called an *obligation*, at the next step ($\bigcirc p_T$) or before the session ends ($\Diamond p_T$). Here $p_T \in V$ is the predicate that holds exactly at a call of T. When its arguments are determined, δ returns the obligated call $(T, \beta_T(\kappa, \rho))$ when the obligation falls due. Otherwise the model is consulted, with T among the admissible tools.

Contract-based decision policy. Write $\alpha(s) := \alpha(\tau)$ and $\delta(s) := \delta(\tau)$ for a state s with trace τ, and let $a_L \sim \pi_{\mathrm{LLM}}(\cdot \mid s, \alpha(s))$ be the call the model proposes when it is consulted at s. The library $\mathcal{C}$ induces the *contract-based decision policy* $U_{\mathcal{C}} : S \to \Delta(\Sigma_c \cup \{\varepsilon\})$,

$$U_{\mathcal{C}}(s) = \begin{cases} \delta(s), & \delta(s) \neq \perp, \\ a_L, & \delta(s) = \perp \ \wedge \ \sigma(\tau\, a_L) = \mathsf{pass}, \\ \varepsilon, & \delta(s) = \perp \ \wedge \ \sigma(\tau\, a_L) = \mathsf{block}, \end{cases} \tag{10}$$

where ε is the empty sequence, so the step adds no event to the trace. Under $\mathcal{C}$ the agent executes $U_{\mathcal{C}}(s)$ in place of the draw from $\pi_{\mathrm{LLM}}(\cdot \mid s, \mathcal{T})$. The first case consumes no model call, and the third commits no event, so s is unchanged and the model receives feedback on the block. When the model proposes several calls, $U_{\mathcal{C}}$ applies to each in order. Algorithm 1 (Appendix E) lists the loop.

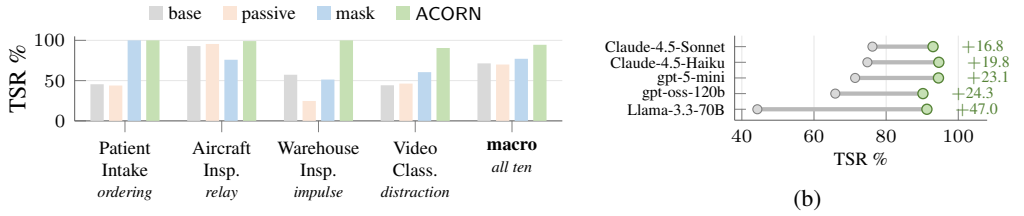
4 Experiments

ACORN is implemented in Python 3.13, with a Z3 checker (de Moura & Bjørner, 2008) for satisfiability checking. Each library is extracted by a language model from the benchmark's SOP document or policy prose, given the atomic propositions (Xiao & Nuzzo, 2026), then reviewed by a human and checked conflict-free before deployment. Appendix B classifies the provenance of the rules and checks the data-derived ones on held-out tasks.

Table 1: SOP-Bench with `gpt-5-mini`. TSR per condition, clean rate (100% under every controller), LLM calls per task, share of tool calls executed without the LLM, and cost change.

		TSR (%) ↑				**Clean** (%) ↑	**LLM calls** per task ↓		**Tool calls** w/o LLM (%) ↑	**Cost** change (%) ↓
Domain	n	base	passive	mask	ACORN	base	base	ACORN	ACORN	ACORN
Dangerous Goods	274	84.7	85.4	81.8	**100.0**	100	2.98	0.99	50	−70
Customer Service	156	68.6	69.9	69.9	**100.0**	73	7.28	5.44	16	−46
Patient Intake	66	45.5	43.9	**100.0**	**100.0**	98	2.94	3.73	21	+13
Know Your Business	90	50.0	50.0	51.1	**55.6**	81	8.24	1.00	89	−88
Aircraft Inspection	112	92.9	95.5	75.9	**99.1**	92	6.41	4.19	19	−39
Warehouse Insp.	150	57.3	24.7	51.3	**100.0**	16	5.61	3.90	20	−50
Email Intent	186	92.5	92.5	93.5	**100.0**	99	4.60	1.65	38	−64
Content Flagging	168	**100.0**	**100.0**	98.2	**100.0**	100	5.81	3.00	25	−57
Video Annotation	125	78.4	92.0	88.8	**100.0**	72	9.79	4.78	17	−66
Video Classification	147	44.2	46.3	60.5	**90.5**†	56	10.22	4.12	18	−75
Macro mean		71.4	70.0	77.1	**94.5**	78.7	6.39	3.28	31	−62

†Includes a threshold the SOP leaves unstated, set from the labels, without which this cell is 56.5 and the macro 91.1 (Appendix B).



(a) (b)

Figure 4: (a) TSR by condition on four domains and the macro (`gpt-5-mini`). (b) Macro TSR from baseline to ACORN on five models with the same contract libraries.

Benchmarks. *SOP-Bench* (Nandi et al., 2025), a benchmark of industrial SOPs released by Amazon, tests whether an agent follows a written procedure. We use the ten domains of its first release to measure task success, compliance, and cost, and to attribute each gain to one mechanism. τ^2*-bench* (Barres et al., 2025) pairs the agent with a simulated user under a prose policy and scores outcomes only. We use its retail and airline domains, where only the agent calls tools, to measure what enforcement costs in conversation and what an outcome metric hides.

4.1 Compliance, Success, and Cost on SOP-Bench

We ran all 1,474 tasks, each one benchmark instance, under four conditions that add one mechanism at a time: baseline (no controller), passive (verification only), mask (masking and verification), and ACORN. An *audit* replays the same library over the committed trace of every finished session, and a task is *procedurally clean* when the audit finds no violation and no pending obligation (Table 1). ACORN is procedurally clean on all ten domains, where the baseline ranges from 16% to 100%, and so are passive and mask, so the compliance guarantee follows from σ alone. Task success rate (TSR), the share of tasks whose report matches the labels, rises from 71.4% to 94.5% macro at roughly half the model calls, because 31% of executed actions never reach the model, and the estimated cost falls by 62%. A macro figure is the unweighted mean over the ten domains. With ReAct, ACORN reaches 93.5 TSR and no violation, versus 72.0 for ReAct alone (Appendix C).

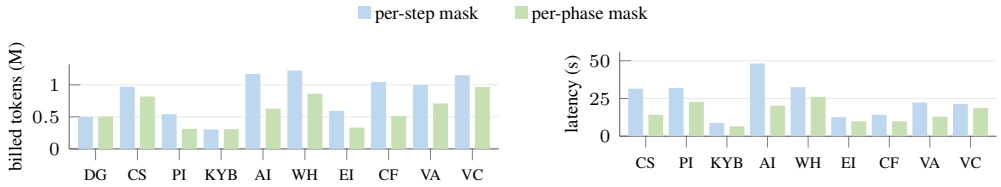
Each mechanism repairs a different failure (Figure 4a). Masking alone raises Patient Intake to 100%, where the model calls the right tools in the wrong order, and adds 10 to 16 pp on the two video domains, whose inventories hide 6 useful tools among 26. In two cells a mechanism lowers TSR. On Aircraft Inspection, masking falls below the baseline, 75.9 against 92.9: the report must repeat values whose vocabulary sits in the schemas of tools the mask has already removed, and the argument map restores the cell by determining the report fields. On Warehouse, passive falls 32.6 pp below the baseline, because its repeated blocks leave 27% of tasks without a report. The two lowest ACORN cells depend on semantic judgments. Know Your Business asks for a judgment that the SOP leaves to human experience, and the escalated cases of Video Classification depend on free-text notes that no argument map determines. Their TSR changes with the backbone.

A library refers only to interaction predicates, so we ran the same ten libraries unedited on four further backbones (Figure 4b). Across the 50 model-by-domain cells, the baseline's procedurally

Table 2: τ^2-bench with a `gpt-4.1-mini` agent and a `gpt-4.1` user simulator. Native $\texttt{pass}^k$ and the audit against the policy contracts (17 retail, 14 airline).

	Outcome ↑		Policy-grounded audit			
Condition	$\texttt{pass}^1$	$\texttt{pass}^4$	viol. rate ↓	blind spot ↓	clean^4 ↑	joint^4 ↑
τ^2-retail, 456 simulations per condition						
Official `LLMAgent`	**63.6**	31.6	37.3	35.9	19.3	8.8
shell (ours, empty library)	62.5	37.7	**0.0**	**0.0**	**100.0**	37.7
passive (verification only)	61.2	31.6	**0.0**	**0.0**	**100.0**	31.6
mask (no fast-forward)	62.1	36.0	**0.0**	**0.0**	**100.0**	36.0
ACORN* (17 grounded contracts)	63.4	**39.5**	**0.0**	**0.0**	**100.0**	**39.5**
ACORN* (full 53 contracts)	55.5	26.3	**0.0**	**0.0**	**100.0**	26.3
τ^2-airline, 200 simulations per condition						
Official `LLMAgent`	40.5	20.0	55.0	34.6	16.0	8.0
shell (ours, empty library)	43.5	22.0	21.0	**0.0**	70.0	22.0
passive (verification only)	54.5	32.0	**0.0**	**0.0**	**100.0**	32.0
mask (no fast-forward)	58.5	40.0	**0.0**	**0.0**	**100.0**	40.0
ACORN* (14 grounded contracts)	**62.0**	**42.0**	**0.0**	**0.0**	**100.0**	**42.0**
ACORN* (full 20 contracts)	56.0	38.0	**0.0**	**0.0**	**100.0**	38.0

*Fast-forward never fires here, since neither policy determines a single call, so ACORN's results come from verification and the mask alone.



(a) (b)

Figure 5: Per-domain billed tokens (a) and latency (b) when the tool schemas are recomputed every step and when they are held fixed over each stage of the task.

clean rate covers the whole range from none to all tasks, while ACORN's is at least 99% in every cell. ACORN commits no violation in any cell, and its few unclean tasks ended in an API error. Task success improves in 48 of the 50 cells, by more on weaker backbones (from $+47.0$ to $+16.8$ pp), and the ACORN macro stays within four points across backbones (Appendix C).

4.2 Enforcing a Conversational Policy on τ^2-bench

We ran six conditions on each of τ^2-retail and τ^2-airline: the official `LLMAgent`, a *shell* that runs our protocol with an empty library, and the four conditions of §4.1 (setup and baseline parity in Appendix A). We audited every simulation against the contracts written from the policy text (Table 2). The native $\texttt{pass}^k$ scores only the final database state, so the table also reports the violation rate, the *blind spot* of reward-passing simulations that violate a contract, and joint^4, the $\texttt{pass}^4$ restricted to simulations clean in all four trials. The official agent breaks a policy rule in more than a third of the simulations the benchmark scores as successes. Every ACORN success is compliant, so its joint^4 is 4.5 times the official agent's on retail and 5.3 times on airline. On airline, ACORN adds $+21.5$ pp of $\texttt{pass}^1$ over the official agent, $[12.5, 30.5]$ under a paired bootstrap, of which the shell alone gains 3.0. Verification alone accounts for 14.0 of these points, and the mask adds 7.5 more. Half of the gain comes from the cancellation contract, which blocks cancellations the policy forbids but the API does not check. On retail, enforcing the 17 policy contracts leaves $\texttt{pass}^1$ unchanged and gives the best $\texttt{pass}^4$ of any condition.

4.3 Mask Granularity and Token Cost

ACORN is the cheapest condition in nine of ten domains, and in ten of ten when priced per successful task. Masking halves the token bill on its own, and fast-forward removes a further 29% by skipping the model calls of determined steps. The controller takes at most 10^{-3} of wall-clock time, and the residual policy cache of §3.3 answers 44% of mask probes on SOP-Bench and 85% on τ^2 (Appendix C). Masking also has a cost. Recomputing $\mathcal{T}_{\text{adm}}(\tau)$ at every step rewrites the tool schemas in the prompt, so the provider's prefix cache misses and every call is billed at the uncached rate, a penalty the numbers above include in full. Holding the tool schemas fixed over each stage of the task, and leaving σ to block the calls this admits beyond $\mathcal{T}_{\text{adm}}(\tau)$, recovers the cache at 30% fewer billed tokens and $1.6\times$ lower latency, up to $2.4\times$ (Figure 5). Task success agrees exactly on

seven of the ten domains and differs by 0.2 pp of macro TSR, and verification is identical in both settings, so the two settings differ in cost and latency and give the same compliance. A third setting leaves the tool schemas untouched and names the admissible tools in a message instead. On Patient Intake it costs 2.3 extra model calls per task and doubles the latency (Appendix D), so the mask works by removing options from the model's interface.

5 Related Work

Guardrails and runtime enforcement. Guardrails block proposed actions or messages that break rules (Rebedea et al., 2023; Inan et al., 2023; Wang et al., 2026), as does our passive condition. Learned or LLM-judged shields are probabilistic (Chen et al., 2025; Wang et al., 2025; Bhardwaj, 2026), auditors and LLM judges grade trajectories post hoc (Zheng et al., 2023; Sharma et al., 2026; Wen et al., 2026), logic-based enforcers check temporal constraints (Kamath et al., 2025; Miculicich et al., 2025; Elkoussy & Perez, 2026; Xiao & Nuzzo, 2026), and privilege control and information-flow designs restrict tools (Shi et al., 2025a; Debenedetti et al., 2025). All of them leave the model in control of every step, so a blocked proposal is often followed by another at the same state (Zhou et al., 2026; Ray, 2026). ACORN inherits the A/G contracts of ContrAgent and adds the mask and fast-forward. Edit automata insert only repairs (Ligatti et al., 2005).

Training-time alignment. Alignment from human feedback makes undesirable behavior unlikely but gives no guarantee for a given call (Casper et al., 2023). Models exploit flaws in their reward, known as reward hacking (Amodei et al., 2016; Skalse et al., 2022), and game their graders (Baker et al., 2025). Aligned models also take actions they recognize as prohibited when these serve an assigned goal (Lynch et al., 2025), and comply selectively when they infer they are being trained (Greenblatt et al., 2024). ACORN's trace-level guarantee holds for any backbone.

Workflow-guided agents and tool selection. Workflow-guided agents traverse a graph or program that encodes the procedure, such as a state machine of prompts and tool calls (Wu et al., 2024), a procedure description language (Shi et al., 2025b), a policy graph built by an LLM (Luo et al., 2026; Kang et al., 2026), an SOP decision graph that exposes only the actions on the current branch (Ye et al., 2025), or a code blueprint that calls the model only for bounded subtasks (Qiu et al., 2025), so the admissible orders of steps are fixed when it is written. Tool-selection methods narrow the tools at each step by retrieval (Franko, 2025) or by planning over preconditions and effects (Babu & Iyer, 2026), without a compliance guarantee. ACORN keeps each requirement as a separate contract, from which it derives the mask, the determined steps, and the check on every call, and its cache resembles cross-history reuse in constrained sampling (Shihab et al., 2026).

Decoding-time and model-internal control. Constrained decoding restricts tokens to a grammar (Willard & Louf, 2023), and jump-forward decoding skips the tokens that the grammar determines (Zheng et al., 2024). Fast-forward is its counterpart for actions. Activation steering and barrier-function filters act on model internals (Turner et al., 2023; Rimsky et al., 2024; Zou et al., 2023; Lee et al., 2025; Miyaoka & Inoue, 2025; Tran et al., 2026), which needs white-box access and generalizes unevenly (Tan et al., 2024).

6 Discussion

We presented ACORN, an agent control framework that compiles contract libraries into runtime control. It exposes only the tools the contracts admit, fast-forwards the calls they determine, and verifies every call. On SOP-Bench it commits no procedural violation and raises task success at lower cost across five backbones, and on τ^2-bench it also commits no policy violation and improves task success. ACORN guarantees compliance with its contract library, not with the procedure behind it. Translating the SOP into contracts with a language model and mining further contracts from traces can both miss or distort rules, which then pass into enforcement unchanged. The contracts also refer only to tool calls, and messages exchanged with the user admit no deterministic evaluation, so a rule about them holds only when a tool call reflects it. Future work includes extending the mask from tools to arguments, modeling data flow from tool results into later calls, and adding predicates over the agent's inputs and outputs.

AI USE STATEMENT

In this work, we used generative AI tools to help interpret experimental results and to provide feedback on the experimental design, and all interpretations and design decisions were made by the authors. We have not used generative AI tools to generate synthetic data, develop theoretical models or conceptual frameworks, formulate mathematical claims, prove them, propose hypotheses, or implement methods, and translation, dataset cleaning, and qualitative data analysis are not applicable to this work. The language model that drafts each contract library from a procedure document is a component of the method and is described in §4. Additionally, we used generative AI tools to draft and edit parts of the paper, suggest its structure, search for and identify relevant literature, create and edit code, and format references. We have reviewed all AI-assisted work: the authors revised all text and verified every cited work, AI-edited code was tested against the framework's test suite, and every reported number was checked against the results ledger. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work builds enforcement infrastructure intended to make LLM agents follow written procedures, and its primary effect is to reduce unauthorized or non-compliant actions. All experiments run against public benchmarks with simulated tools and synthetic data. No human subjects, personal data, or live systems are involved. The same masking machinery could in principle encode restrictive policies we would consider harmful, but the contract library is an explicit, auditable artifact, which makes such policies inspectable rather than implicit in model weights.

REPRODUCIBILITY STATEMENT

The framework, the per-domain benchmark adapters (including the documented boundary notes and the tests that pin label-noise tasks), the contract libraries, and the evaluation scripts for both benchmarks will be released. Appendix E gives the runtime loop as pseudocode. Appendix A gives the statistical procedure, the τ^2-bench setup, and the steps taken for baseline parity. Appendix B classifies the provenance of every rule and checks the authoring decisions on held-out tasks. Every number in the paper is transcribed from a single results ledger generated by the evaluation scripts, and macro rows are unweighted means over the per-domain cells shown.

REFERENCES

Luca Aceto, Ian Cassar, Adrian Francalanza, and Anna Ingólfsdóttir. On bidirectional runtime enforcement. In *Formal Techniques for Distributed Objects, Components, and Systems (FORTE)*, LNCS. Springer, 2021.

Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. 2016. arXiv:1606.06565.

Rahul Suresh Babu and Laxmipriya Ganesh Iyer. ToolChoiceConfusion: Causal minimal tool filtering for reliable LLM agents, 2026. URL https://arxiv.org/abs/2606.06284. arXiv preprint.

Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, Cambridge, MA, 2008.

Bowen Baker, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y. Guan, Aleksander Madry, Wojciech Zaremba, Jakub Pachocki, and David Farhi. Monitoring reasoning models for misbehavior and the risks of promoting obfuscation, 2025. arXiv preprint arXiv:2503.11926.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment, 2025. URL https://arxiv.org/abs/2506.07982. arXiv preprint.

Andreas Bauer, Martin Leucker, and Christian Schallhart. Monitoring of real-time properties. In *Proc. Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 4337 of *Lecture Notes in Computer Science*, pp. 260–272. Springer, 2006. doi: 10.1007/11944836_25.

Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(4):14:1–14:64, 2011. doi: 10.1145/2000799.2000800.

Albert Benveniste, Benoît Caillaud, Alberto Ferrari, Leonardo Mangeruca, Roberto Passerone, and Christos Sofronis. Multiple viewpoint contract-based specification and design. In *Formal Methods for Components and Objects (FMCO)*, volume 5382 of *LNCS*, pp. 200–225. Springer, 2008.

Albert Benveniste, Benoît Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, Philipp Reinkemeier, Alberto Sangiovanni-Vincentelli, Werner Damm, Thomas A. Henzinger, and Kim G. Larsen. Contracts for system design. *Foundations and Trends in Electronic Design Automation*, 12(2–3):124–400, 2018. doi: 10.1561/1000000053.

Varun Pratap Bhardwaj. Agent behavioral contracts: Formal specification and runtime enforcement for reliable autonomous AI agents, 2026. URL https://arxiv.org/abs/2602.22302. arXiv preprint.

Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, Jérémy Scheurer, Javier Rando, Rachel Freedman, Tomasz Korbak, David Lindner, Pedro Freire, et al. Open problems and fundamental limitations of reinforcement learning from human feedback. *Transactions on Machine Learning Research*, 2023. arXiv:2307.15217.

Zhaorun Chen, Mintong Kang, and Bo Li. ShieldAgent: Shielding agents via verifiable safety policy reasoning. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025. URL https://arxiv.org/abs/2503.22738.

Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 854–860, 2013. URL https://www.ijcai.org/Proceedings/13/Papers/132.pdf.

Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4963 of *Lecture Notes in Computer Science*, pp. 337–340. Springer, 2008. doi: 10.1007/978-3-540-78800-3_24.

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating prompt injections by design, 2025. URL https://arxiv.org/abs/2503.18813. arXiv preprint.

Laïla Elkoussy and Julien Perez. AgentLTL: A trace-verification framework for measuring, enforcing, and training procedural compliance in tool-using LLM agents, 2026. URL https://arxiv.org/abs/2607.02599. arXiv preprint.

Paolo Felli, Marco Montali, Fabio Patrizi, and Sarah Winkler. Monitoring arithmetic temporal properties on finite traces. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI)*, pp. 6346–6354. AAAI Press, 2023. URL https://doi.org/10.1609/aaai.v37i5.25781.

Uria Franko. Dynamic system instructions and tool exposure for efficient agentic LLMs, 2025. URL https://arxiv.org/abs/2602.17046. arXiv preprint.

Luca Geatti, Alessandro Gianola, and Nicola Gigante. Linear temporal logic modulo theories over finite traces. In *Proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2641–2647. ijcai.org, 2022. URL https://doi.org/10.24963/ijcai.2022/366.

Ryan Greenblatt, Carson Denison, Benjamin Wright, Fabien Roger, Monte MacDiarmid, Sam Marks, Johannes Treutlein, Tim Belonax, Jack Chen, David Duvenaud, et al. Alignment faking in large language models, 2024. arXiv preprint arXiv:2412.14093.

Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, and Madian Khabsa. Llama Guard: LLM-based input-output safeguard for human-AI conversations, 2023. URL https://arxiv.org/abs/2312.06674. arXiv preprint.

Adharsh Kamath, Sishen Zhang, Calvin Xu, Shubham Ugare, Gagandeep Singh, and Sasa Misailovic. Enforcing temporal constraints for LLM agents, 2025. URL https://arxiv.org/abs/2512.23738. arXiv preprint.

Seongjae Kang, Taehyung Yu, and Sung Ju Hwang. PolicyGuide: From guarding one action to guiding the whole workflow for policy-compliant LLM agents. *arXiv preprint arXiv:2608.19861*, 2026.

Bruce W. Lee, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Erik Miehling, Pierre Dognin, Manish Nagireddy, and Amit Dhurandhar. Programming refusal with conditional activation steering. In *International Conference on Learning Representations (ICLR)*, 2025.

Zekun Li, Shinda Huang, Jiangtian Wang, Nathan Zhang, Antonis Antoniades, Wenyue Hua, Kaijie Zhu, Sirui Zeng, Chi Wang, William Yang Wang, and Xifeng Yan. SOPBench: Evaluating language agents at following standard operating procedures and constraints, 2025. URL https://arxiv.org/abs/2503.08669. arXiv preprint.

Jay Ligatti, Lujo Bauer, and David Walker. Edit automata: Enforcement mechanisms for run-time security policies. *International Journal of Information Security*, 4(1–2):2–16, 2005. doi: 10.1007/s10207-004-0046-8.

Mengxi Luo, Changjia Chen, An Cao, Zirong Huang, and Wanyi Dai. STAGE: Stateful translation to agentic graph execution with policy-scoped context and deterministic control. *arXiv preprint arXiv:2608.22538*, 2026.

Aengus Lynch, Benjamin Wright, Caleb Larson, Kevin K. Troy, Stuart J. Ritchie, Sören Mindermann, Ethan Perez, and Evan Hubinger. Agentic misalignment: How LLMs could be insider threats, 2025. arXiv preprint arXiv:2510.05179.

Fabrizio Maria Maggi, Michael Westergaard, Marco Montali, and Wil M. P. van der Aalst. Runtime verification of LTL-based declarative process models. In *Runtime Verification: Revised Selected Papers of RV 2011*, volume 7186 of *Lecture Notes in Computer Science*, pp. 131–146. Springer, 2012. doi: 10.1007/978-3-642-29860-8_11.

Lesly Miculicich, Mihir Parmar, Hamid Palangi, Krishnamurthy Dj Dvijotham, Mirko Montanari, Tomas Pfister, and Long T. Le. VeriGuard: Enhancing LLM agent safety via verified code generation, 2025. URL https://arxiv.org/abs/2510.05156. arXiv preprint.

Yuya Miyaoka and Masaki Inoue. Control barrier function for aligning large language models. *arXiv preprint arXiv:2511.03121*, 2025.

Subhrangshu Nandi, Arghya Datta, Rohith Nama, Udita Patel, Nikhil Vichare, Indranil Bhattacharya, Prince Grover, Shivam Asija, Giuseppe Carenini, Wei Zhang, et al. SOP-Bench: Complex industrial SOPs for evaluating LLM agents, 2025. v1, 9 Jun 2025; revised as v2, 23 Feb 2026.

Libin Qiu, Yuhang Ye, Zhirong Gao, Xide Zou, Junfu Chen, Ziming Gui, Weizhi Huang, Xiaobo Xue, Wenkai Qiu, and Kun Zhao. Blueprint first, model second: A framework for deterministic LLM workflow, 2025. URL https://arxiv.org/abs/2508.02721. arXiv preprint.

Shawn Ray. What can be enforced? A theory of certified runtime safety for tool-using agents, 2026. URL https://arxiv.org/abs/2607.22868. arXiv preprint.

Traian Rebedea, Razvan Dinu, Makesh Sreedhar, Christopher Parisien, and Jonathan Cohen. NeMo guardrails: A toolkit for controllable and safe LLM applications with programmable rails. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP): System Demonstrations*, 2023.

Nina Rimsky, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Matt Turner. Steering Llama 2 via contrastive activation addition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Grigore Roşu and Klaus Havelund. Rewriting-based techniques for runtime verification. *Automated Software Engineering*, 12(2):151–197, 2005.

Reshabh K. Sharma, Shraddha Barke, and Benjamin Zorn. Willful disobedience: Automatically detecting failures in agentic traces. In *Proceedings of the ACM Conference on AI and Agentic Systems (CAIS)*, 2026. URL https://arxiv.org/abs/2603.23806.

Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Securing AI agents with privilege control, 2025a. URL https://arxiv.org/abs/2504.11703. arXiv preprint.

Yuchen Shi, Siqi Cai, Zihan Xu, Yuei Qin, Gang Li, Hang Shao, Jiawei Chen, Deqing Yang, Ke Li, and Xing Sun. FlowAgent: Achieving compliance and flexibility for workflow agents. *arXiv preprint arXiv:2502.14345*, 2025b.

Ibne Farabi Shihab, Md Najmus Swaqeeb, and Abu Sa-Adat Mohamed Moon-Im Al Ahsan. Stateful CARS: Exact cross-history reuse for policy-constrained LLM agents, 2026. URL https://arxiv.org/abs/2608.08282. arXiv preprint.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Daniel Tan, David Chanin, Aengus Lynch, Dimitrios Kanoulas, Brooks Paige, Adria Garriga-Alonso, and Robert Kirk. Analyzing the generalization and reliability of steering vectors. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Thanh Q. Tran, Arun Verma, Kiwan Wong, Bryan Kian Hsiang Low, Daniela Rus, and Wei Xiao. BarrierSteer: LLM safety via learning barrier steering. *arXiv preprint arXiv:2602.20102*, 2026.

Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J. Vazquez, Ulisse Mini, and Monte MacDiarmid. Steering language models with activation engineering, 2023. arXiv preprint arXiv:2308.10248.

Haoyu Wang, Christopher M. Poskitt, Jiali Wei, and Jun Sun. ProbGuard: Proactive runtime monitoring for LLM agent safety via probabilistic prediction, 2025. URL https://arxiv.org/abs/2508.00500. arXiv preprint.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents. In *Proceedings of the 48th IEEE/ACM International Conference on Software Engineering (ICSE)*, 2026. URL https://arxiv.org/abs/2503.18666.

Xiaofei Wen, Wenjie Jacky Mo, Yanan Xie, Peng Qi, and Muhao Chen. Learning efficient guardrails for compliance. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2510.03485.

Brandon T. Willard and Rémi Louf. Efficient guided generation for large language models, 2023. arXiv preprint arXiv:2307.09702.

Yiran Wu, Tianwei Yue, Shaokun Zhang, Chi Wang, and Qingyun Wu. StateFlow: Enhancing LLM task-solving through state-driven workflows. *arXiv preprint arXiv:2403.11322*, 2024.

Yifeng Xiao and Pierluigi Nuzzo. Symbolic temporal supervision of LLM agents using contracts, 2026. URL https://arxiv.org/abs/2609.18128.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023. URL https://arxiv.org/abs/2210.03629.

Anbang Ye, Qianran Ma, Jia Chen, Muqi Li, Tong Li, Fujiao Liu, Siqi Mai, Meichen Lu, Haitao Bao, and Yang You. SOP-Agent: Empower general purpose AI agent with domain-specific SOPs, 2025. URL https://arxiv.org/abs/2501.09316. arXiv preprint.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023. URL https://arxiv.org/abs/2306.05685.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Chenyu Zhou, Qiliang Jiang, Shuning Wu, and Xu Zhou. SiLR: Structure-preserving admission and process reward for LLM tool agents, 2026. URL https://arxiv.org/abs/2609.04629. arXiv preprint.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, et al. Representation engineering: A top-down approach to AI transparency. *arXiv preprint arXiv:2310.01405*, 2023.

A Experimental Details

Statistical uncertainty. Macro 95% confidence intervals come from a bootstrap over tasks (10,000 resamples within each domain, macro taken as the unweighted mean of domain means). For the four Bedrock backbones the bootstrap resamples the recorded per-task results: ACORN vs. baseline macros are $[93.5, 95.8]$ vs. $[72.7, 76.9]$ (Claude-4.5-Haiku), $[88.5, 91.7]$ vs. $[63.6, 68.1]$ (gpt-oss-120b), $[90.0, 92.7]$ vs. $[42.3, 46.3]$ (Llama-3.3-70B), and $[91.8, 94.2]$ vs. $[74.1, 78.3]$ (Claude-4.5-Sonnet). For `gpt-5-mini`, whose per-task results were not archived, it is parametric (per-domain Binomial at the observed rate): $[93.4, 95.7]$ vs. $[69.0, 73.7]$. Every ACORN interval is disjoint from its baseline interval. The tool environments are deterministic, so most cells are single runs. On Dangerous Goods, three trials of 20 tasks give ACORN 100% and the baseline 65–75%.

τ^2-bench. The ACORN agent implements the benchmark's half-duplex agent interface. Prescriptive guarantees narrow the mask to the named tool, liveness guarantees are driven at session end (§3.4), and the obligated tool receives the arguments of the triggering call that match its schema. Simulations lost to infrastructure failures count as failures (at most 3% in any condition). Airline runs use concurrency 4 with 32 model-call retries. Its grounded library holds 12 contracts plus the two cancellation contracts of §4.2, and the full library adds six trace-mined conventions. A paired bootstrap over the 50 airline tasks gives grounded ACORN minus the official agent $+21.5$ pp $[12.5, 30.5]$ on $\texttt{pass}^1$ and $+21.9$ $[8.0, 36.0]$ on $\texttt{pass}^4$. The cancellation contract approximates two inputs: a flight counts as cancelled by the airline only if the agent looked up its status, and a covered reason is a keyword pattern over the user's messages. Our rerun of the official agent matches its published share of violating simulations (55.0% vs. 59.5%).

Baseline parity on τ^2-bench. Before the final comparison we removed four confounds, each of which moved the official agent's behavior: the system-prompt wrapper, sampling temperature (provider default 1.0 vs. the official 0.0), retry policy under rate limits, and tool-schema and message-role fidelity in the adapter. Three runs at intermediate points of this checklist scored $\texttt{pass}^1$ 0.550, 0.561, and 0.555, so the final number is stable rather than tuned.

B Library Provenance and Held-Out Check

Library provenance. Every rule in the ten libraries has one of three provenances. SOP-derived rules formalize the procedure text or the tool schemas, and the compliance guarantee rests on them alone. Data-validated rules are stated by the SOP and were checked against the labeled tasks before being enforced, so the labels can veto such a rule but add nothing to it. Data-derived elements take content from the labels, either to settle an ambiguous SOP clause or to fill in a value the SOP names but never states, such as the Video Classification threshold. They enter argument maps and completeness gates, so they can raise TSR but never affect compliance. Removing the Video Classification threshold and letting the argument map abstain lowers ACORN on that domain from 90.5% to 56.5%, against 44.2% for the baseline, with every task still procedurally clean.

Held-out authoring split. To check that these decisions are not fitted to the evaluated tasks, we split each domain's labeled tasks at random into an authoring half and a held-out half and re-derived every data-dependent decision from the authoring half alone. Every decision re-derives identically, so the held-out half is a fair evaluation set. Table 3 re-scores the four Bedrock backbones on it from the recorded results. Each macro moves by at most 1.4 pp, and the gains are essentially unchanged. The `gpt-5-mini` results were not archived per task.

Table 3: Macro TSR on the full task set and on the held-out half of the 50/50 authoring split (same libraries, same recorded runs).

	ACORN (%)		**baseline** (%)	
model	full	held-out	full	held-out
`claude-4.5-haiku`	94.6	94.2	74.8	74.1
`gpt-oss-120b`	90.2	88.8	65.9	65.6
`llama-3.3-70b`	91.3	90.7	44.3	45.4
`claude-4.5-sonnet`	93.0	92.5	76.2	76.1

Table 4: Matched-model matrix on Bedrock (step masking; ACORN vs. the same model's function-calling baseline; proc-clean by the audit). All four models are complete (10/10 domains). The same contract libraries transfer unchanged.

			TSR (%)		**proc-clean** (%)	
model	**domain**	n	base	ACORN	base	ACORN
`claude-4.5-haiku`	Dangerous Goods	274	81.8	**100.0**	99	**100**
	Customer Service	156	51.3	**100.0**	62	**100**
	Patient Intake	66	92.4	**100.0**	100	**100**
	Know Your Business	90	50.0	**58.9**	94	**100**
	Aircraft Inspection	112	87.5	**100.0**	97	**100**
	Warehouse Pkg Insp.	150	49.3	**98.7**	49	**100**
	Email Intent	186	96.2	**98.4**	100	**100**
	Content Flagging	168	100.0	**100.0**	100	**100**
	Video Annotation	125	84.0	**100.0**	74	**100**
	Video Classification	147	55.8	**90.5**	50	**100**
	macro		74.8	**94.6**		
`claude-4.5-sonnet`	Dangerous Goods	274	83.2	**100.0**	100	**100**
	Customer Service	156	74.4	**98.7**	64	**99**
	Patient Intake	66	98.5	**100.0**	100	**100**
	Know Your Business	90	55.6	**51.1**	98	**100**
	Aircraft Inspection	112	77.7	**100.0**	79	**100**
	Warehouse Pkg Insp.	150	51.3	**99.3**	74	**100**
	Email Intent	186	96.2	**98.4**	99	**100**
	Content Flagging	168	98.2	**98.8**	98	**99**
	Video Annotation	125	77.6	**100.0**	69	**100**
	Video Classification	147	49.7	**83.7**	90	**100**
	macro		76.2	**93.0**		
`gpt-oss-120b`	Dangerous Goods	274	73.4	**100.0**	99	**100**
	Customer Service	156	69.9	**92.9**	69	**100**
	Patient Intake	66	97.0	**100.0**	97	**100**
	Know Your Business	90	56.7	**56.7**	20	**100**
	Aircraft Inspection	112	30.4	**81.2**	25	**100**
	Warehouse Pkg Insp.	150	40.0	**99.3**	11	**100**
	Email Intent	186	95.7	**98.4**	77	**100**
	Content Flagging	168	95.8	**100.0**	93	**100**
	Video Annotation	125	57.6	**84.0**	54	**100**
	Video Classification	147	42.2	**89.1**	12	**100**
	macro		65.9	**90.2**		
`llama-3.3-70b`	Dangerous Goods	274	0.0	**100.0**	98	**100**
	Customer Service	156	20.5	**91.7**	29	**100**
	Patient Intake	66	77.3	**100.0**	50	**100**
	Know Your Business	90	38.9	**52.2**	19	**99**
	Aircraft Inspection	112	1.8	**99.1**	0	**99**
	Warehouse Pkg Insp.	150	31.3	**100.0**	6	**100**
	Email Intent	186	91.9	**81.2**	74	**100**
	Content Flagging	168	100.0	**100.0**	96	**100**
	Video Annotation	125	71.2	**100.0**	0	**100**
	Video Classification	147	10.2	**89.1**	0	**100**
	macro		44.3	**91.3**		

C Cross-Model Notes and Efficiency Accounting

Neural scaffolds. A ReAct condition (same model, tools, and prompts, on `claude-4.5-haiku`) does not provide compliance: its procedure-clean macro is 90.5% against ACORN's 100%. It is not uniformly better than function calling either, and it collapses to 18.0% on the branching Warehouse domain. It does compose with ACORN, which lifts it to 98–100% on three domains and drives committed violations to zero (nine-domain macro: ReAct 72.0 against ACORN 93.5 TSR). The contract layer is therefore orthogonal to the reasoning scaffold, not a competitor to it.

Cells that run against the trend. On Know Your Business the strong models stay flat or drop (Sonnet 55.6→51.1, gpt-oss 56.7→56.7), because its escalation judgment does not depend on the controller (§4.1). Llama drops on Email Intent (91.9→81.2), because the masked condition's strict output schema rejects free-text labels that the baseline parser accepts. Sonnet first exhausted its step budget on Aircraft Inspection and Video Classification by inventing tool names. Feedback that names the admissible tools resolves this, and Table 4 reports the reruns (100.0 and 83.7).

Efficiency accounting. Table 5 reports model-token totals, estimated cost, and mean per-task latency for every (domain, condition) cell. The controller's wall-clock share is at most 10^{-3} everywhere. Under ACORN the residual policy cache answers a mean 44% of mask probes across the nine

SOP-Bench domains where reuse was recorded, from 0% on Email Intent, whose sessions repeat no control state, to 91% on Aircraft Inspection, and 84–85% on τ^2-retail.

Table 5: Per-domain model tokens (millions), estimated cost ($), and mean per-task latency (s) for the four conditions (`gpt-5-mini`); "–" marks cells whose latency was not recorded.

	tokens (M)				**cost** ($)				**latency** (s)			
domain	base	pass.	mask	ACORN	base	pass.	mask	ACORN	base	pass.	mask	ACORN
Dangerous Goods	1.67	1.66	0.99	0.50	1.00	1.00	0.60	0.30	–	16.8	14.4	–
Customer Service	4.02	4.17	2.73	2.16	2.41	2.50	1.64	1.30	25.9	33.4	62.0	31.5
Patient Intake	0.54	0.52	0.73	0.61	0.32	0.31	0.44	0.36	21.9	20.3	38.0	32.0
Know Your Business	2.58	2.71	2.04	0.30	1.55	1.63	1.22	0.18	32.4	38.4	36.2	8.8
Aircraft Inspection	2.33	2.28	1.62	1.41	1.40	1.37	0.97	0.85	35.5	29.7	47.4	48.2
Warehouse Insp.	2.45	2.88	1.46	1.22	1.47	1.73	0.87	0.73	30.2	39.1	36.3	32.5
Email Intent	1.99	1.99	1.04	0.72	1.20	1.19	0.62	0.43	21.9	22.5	19.3	12.6
Content Flagging	2.46	2.46	1.35	1.04	1.48	1.48	0.81	0.63	15.9	16.6	23.4	14.2
Video Annotation	9.67	10.46	3.99	3.26	5.80	6.27	2.39	1.96	30.5	35.0	29.8	22.4
Video Classification	5.29	5.36	1.66	1.31	3.17	3.22	1.00	0.79	29.4	33.4	35.3	21.3
total / macro	33.0	34.5	17.6	12.5	19.80	20.70	10.56	7.53	–	–	–	–

D MASK GRANULARITY MEASUREMENTS

Figure 5 and §4.3 compare three ways of presenting the admissible tools to the model. step recomputes the tool schemas at every step, phase holds them fixed over each stage of the task and leaves σ to block the calls this admits beyond $\mathcal{T}_{\mathrm{adm}}(\tau)$, and hint leaves the schemas untouched and names the admissible tools in a message. Verification is identical in all three, and committed violations are zero in every setting. Table 6 compares step and phase on all ten domains, and Table 7 adds hint on two of them.

Table 6: step vs. phase masking on the ten SOP-Bench domains (ACORN, `gpt-5-mini`): TSR and change in billed tokens (uncached $+\,0.1\times$ cached). Figure 5 plots tokens and latency.

	TSR (%)		**Billed tokens**
Domain	step	phase	change
Dangerous Goods	100.0	100.0	$+1\%$
Customer Service	100.0	100.0	-16%
Patient Intake	100.0	100.0	-42%
Know Your Business	55.6	55.6	$+1\%$
Aircraft Inspection	99.1	100.0	-46%
Warehouse Insp.	100.0	97.3	-30%
Email Intent	100.0	99.5	-44%
Content Flagging	100.0	100.0	-51%
Video Annotation	100.0	100.0	-29%
Video Classification	90.5	90.5	-16%
Total / macro	94.5	94.3	-30%

Table 7: Three-granularity comparison including hint, on one procedural and one conversational domain. phase rows are from the sweep of Table 6.

domain	**granularity**	**TSR** (%)	**latency**	**calls**/task	**billed tok.**
Patient Intake	step	**100.0**	32.0s	3.7	≈541k
Patient Intake	phase	**100.0**	22.8s	4.0	≈313k
Patient Intake	hint	98.5	48.4s	6.0	≈497k
Customer Service	step	**100.0**	31.5s	5.4	≈969k
Customer Service	phase	**100.0**	14.2s	5.4	≈818k
Customer Service	hint	**100.0**	19.8s	5.5	≈947k

E RUNTIME LOOP PSEUDOCODE

Algorithm 1 The ACORN runtime loop (one session)

compile each contract of $\mathcal{C}$ into its DFAs; $\tau \leftarrow \varepsilon$
Exec(a): append a to τ; run the tool; append its return event r to τ if $\sigma(\tau\, r) = \mathsf{pass}$, and suppress r otherwise
while the session has not ended **do**
  $\alpha(\tau) \leftarrow \mathcal{T}_{\mathrm{adm}}(\tau)$ of Eq. (8) {mask}
  if $\delta(\tau) \neq \bot$ and $\sigma(\tau\, \delta(\tau)) = \mathsf{pass}$ **then**
    Exec$(\delta(\tau))$ {fast-forward, Eq. (9); no model call}
  else if $\mathcal{T}_{\mathrm{adm}}(\tau) \neq \emptyset$ **then**
    $y \leftarrow$ the model's response to the transcript, with the tools $\mathcal{T}_{\mathrm{adm}}(\tau)$
    if y proposes no call **then**
      end the session if it may stop at τ (Eq. (4) with empty τ'); otherwise return the pending obligations to the model
    end if
    for all calls a proposed in y, in order **do**
      if $\sigma(\tau\, a) = \mathsf{pass}$: **Exec**$(a)$ {verify, §3.2}
      else: return feedback on the block: the contracts a would violate and the tools that supply a missing prerequisite
    end for
  else
    end the session
  end if
end while
report the session as complete if it may stop at τ, and otherwise report its pending obligations