Formal-State-Augmented Language Models: Invariant-Preserving Execution with External Formal State

Adrian Diamond
AADSystems Research
adrian@AADSystems.com

November 1, 2026

Abstract

How can a probabilistic language model act within a computational system without surrendering deterministic control over consequential state changes?

This paper addresses that question through a formal-state-augmented architecture in which language models generate candidate actions but possess no independent authority to modify the system's state. Instead, an external formal executor governs execution through explicit transition semantics, admissibility conditions, and state invariants.

We develop an operational semantics for this separation of probabilistic proposal generation from deterministic state execution. We establish that, under sound checking and non-bypassable execution assumptions, the system preserves its specified invariants across arbitrary finite sequences of model-generated proposals, independently of the language model's behavior. We further characterize conditions under which incomplete state representations cannot preserve action-admissibility decisions.

The mathematical framework is accompanied by an executable reference monitor, automated tests, bounded evaluations, and an interactive demonstration. A language-model integration provides a basis for evaluating proposed actions against independently enforced formal constraints.

The central contribution is an explicit architecture for combining probabilistic intelligence with deterministic control over authorized state transitions. Its guarantees concern formally specified properties of execution, not the general truthfulness or reliability of language-model outputs.

Contents

1 Introduction **3**

2 Formal-State-Augmented Architecture **3**
2.1 State, proposals, and authority 3
2.2 Operational boundary 4

3 Invariant Preservation and Its Limits **4**

4 State Rendering and the Information Boundary **5**
4.1 Indistinguishability of admissibility 5
4.2 Observation preservation is a separate question 6

5 A Worked Quota-and-Permission Example **6**

6 Relationship to Existing Methods and Companion Work **7**

7 Executable Reference Monitor and Bounded Evaluation **7**
7.1 Implementation boundary 7
7.2 Exhaustive single-step comparison 8
7.3 Bounded traces, parser behavior, and state rendering 8
7.4 What remains untested 8

8 Connected-Model Prototype and Evaluation Protocol **9**
8.1 Separating proposed actions from committed transitions 9
8.2 Live evaluation design and evidentiary boundary 9

9 Scope, Limitations, and Open Questions **9**

10 Conclusion **10**

1 Introduction

Language models can generate useful candidate actions, but a generated proposal is not itself an authorization to update a persistent system state. In applications involving resource limits, access permissions, or other operational constraints, an action may be syntactically plausible while violating a requirement of the system in which it is executed. The central design problem is therefore not simply how to make model outputs more likely to conform to instructions, but how to separate probabilistic proposal generation from authoritative state transitions.

Architectural principle. We consider a model that observes an input x and a rendering $r(s)$ of a formally represented state s, then proposes a structured action a according to a distribution $\pi_\theta(a \mid x, r(s))$. A separate executor holds the authoritative state and is solely responsible for committing updates. It accepts a proposal only when the transition is defined, a specified guard permits it, and the resulting state satisfies an explicit invariant. This separation connects language-model architectures (Vaswani et al. 2017) with established methods for reasoning about program invariants (Hoare 1969); it does not, by itself, establish a new invariant principle.

Central claim. If the initial formal state satisfies the invariant, every state update is mediated by a non-bypassable executor, and that executor correctly enforces the specified checks, then all states reached by any finite sequence of proposals satisfy the invariant. The guarantee concerns *committed formal states*, not the factual accuracy, helpfulness, security, or appropriateness of model-generated content. It also depends on whether the formal specifications actually capture the constraints the application needs.

Contributions and scope. This paper (i) gives a minimal state-and-action semantics for the proposer–executor boundary; (ii) proves conditional invariant preservation; (iii) characterizes when state rendering retains enough information to determine action admissibility; and (iv) presents an example separating authorization from successor-state validity. In particular, the rendering shown to a model may omit distinctions retained by the authoritative state. The executor must therefore make decisions from that state and the formal transition rules, not from the model's reconstruction of them.

The paper also includes a reference implementation and a bounded, scripted evaluation of its defined transition semantics. The reported scripted measurements are not live-model evidence; an optional connected-model adapter is included for subsequent experimentation. The prototype is not a verified compiler or a production security monitor. Its preservation proof and representation results employ elementary arguments already familiar in verification and abstraction; the present study is best understood as a precise, reproducible reference construction rather than a demonstrated improvement over existing agent-safety systems. The companion manuscript, *Proof-Carrying State Transitions for Reliable AI Systems* (R16) (Diamond 2026), addresses the distinct question of independently checkable certificates for proposed transitions.

2 Formal-State-Augmented Architecture

2.1 State, proposals, and authority

Definition 2.1 (State-transition specification)**.** *A* state-transition specification *is a tuple*

$$\mathfrak{F} = (\mathcal{S}, \mathcal{A}, U, G, \mathcal{I})$$

where $\mathcal{S}$ is a set of authoritative states, $\mathcal{A}$ is a set of structured actions, $U : \mathcal{S} \times \mathcal{A} \rightharpoonup \mathcal{S}$ is a partial transition function, $G : \mathcal{S} \times \mathcal{A} \to \{\mathsf{true}, \mathsf{false}\}$ is a guard, and $\mathcal{I} : \mathcal{S} \to \{\mathsf{true}, \mathsf{false}\}$ is a state invariant. We write $U(s, a) \downarrow$ when the update is defined.

The different components have distinct responsibilities. Definedness establishes whether an action has a specified successor; the guard expresses authorization or transition-specific policy; the invariant expresses a property required of committed states. These conditions need not be logically equivalent. In particular, a successor can satisfy $\mathcal{I}$ even when the action producing it is unauthorized.

Let X denote external inputs and R representations supplied to a model. A rendering map $r : \mathcal{S} \to R$ may discard information. A proposal mechanism with parameters θ supplies a distribution $\pi_\theta(\cdot \mid x, r(s))$ over actions. The only requirement in our safety result is that the proposal passed to the executor is an element of $\mathcal{A}$; we make no distributional assumptions. A parser that converts raw model text to $\mathcal{A}$ is a separate interface. Invalid parses are rejected without modifying state.

Definition 2.2 (Admissibility and monitored step)**.** *For $s \in \mathcal{S}$ and $a \in \mathcal{A}$, define the admissibility predicate*

$$D(s, a) = \begin{cases} \mathsf{true}, & U(s, a) \downarrow,\ G(s, a) = \mathsf{true},\ \text{and } \mathcal{I}(U(s, a)) = \mathsf{true}, \\ \mathsf{false}, & \text{otherwise.} \end{cases}$$

The monitored executor is the total function

$$M : \mathcal{S} \times \mathcal{A} \longrightarrow \mathcal{S} \times \{\mathsf{accept}, \mathsf{reject}\}, \qquad M(s, a) = \begin{cases} (U(s, a), \mathsf{accept}), & D(s, a) = \mathsf{true}, \\ (s, \mathsf{reject}), & D(s, a) = \mathsf{false}. \end{cases}$$

The expression $U(s, a)$ in the first branch is meaningful because its definedness is part of $D(s, a)$.

A proposed successor described in natural language is not a committed successor. Only M may update the authoritative state. A refusal may still produce an explanatory response, but this response has no state-changing effect under the specified semantics.

2.2 Operational boundary

An execution consists of states $s_0, s_1, \ldots$ and a sequence of proposals $a_0, a_1, \ldots$, where

$$M(s_i, a_i) = (s_{i+1}, d_i), \qquad d_i \in \{\mathsf{accept}, \mathsf{reject}\}.$$

The proposal a_i may depend on all previous messages, rendered states, responses, and decisions. Thus the model may be adaptive. A finite execution trace records (s_i, a_i, d_i, s_{i+1}) for each step; an implementation might store state identifiers or cryptographic commitments rather than full sensitive states. This is a specification of what must be recorded, not an implemented audit mechanism.

For this model to describe an actual system, the executor must have exclusive commit authority, the checker must evaluate the actual current state, and the check and commit must not be separated by an uncontrolled intervening update. Concurrent systems therefore need an appropriate atomic transaction or linearization discipline. The formal safety result below is conditional on this boundary.

3 Invariant Preservation and Its Limits

Theorem 3.1 (Safety of monitored execution)**.** *Suppose $\mathcal{I}(s_0) = \mathsf{true}$. For every finite sequence of proposals $a_0, \ldots, a_{n-1}$, possibly chosen adaptively or probabilistically, let the states $s_1, \ldots, s_n$ be determined by the monitored transition M. Then*

$$\forall i \in \{0, \ldots, n\}, \qquad \mathcal{I}(s_i) = \mathsf{true}.$$

Proof. We argue by induction on i. The base case is the hypothesis. Assume $\mathcal{I}(s_i) = \mathsf{true}$. If M rejects a_i, then $s_{i+1} = s_i$ and the invariant persists. If M accepts it, then $s_{i+1} = U(s_i, a_i)$, and the definition of admissibility entails $\mathcal{I}(s_{i+1}) = \mathsf{true}$. The induction applies regardless of how the proposal was selected. □

Remark 3.2 (What the theorem does and does not prove)**.** *The conclusion is* pathwise*: every sequence of actions processed under the stipulated transition semantics is safe with respect to the stated invariant, not merely safe with high probability. This is a standard inductive preservation argument. It is not a theorem that the language model is truthful, that $\mathcal{I}$ captures every real-world requirement, or that the implementation faithfully realizes M. It also says nothing about progress: a monitor that rejects every proposal preserves safety.*

Proposition 3.3 (Composition of monitored steps)**.** *Let $M_a(s)$ denote the state component of $M(s, a)$. For every finite sequence $a_1, \ldots, a_n$, the composed map $M_{a_n} \circ \cdots \circ M_{a_1}$ sends invariant states to invariant states.*

Proof. By Theorem 3.1, or by repeated application of the single-step preservation property. No closure property of the proposal distribution is required. □

The proposition does not imply that all reasonable sequences of actions are reachable. To establish liveness or nontrivial completeness, one must separately study the guards, the supported transition language, and whether valid actions can be proposed.

4 State Rendering and the Information Boundary

4.1 Indistinguishability of admissibility

A language model can be supplied with a presentation $r(s)$ of a state without having access to the state itself. The question is not whether r is injective, but whether it preserves distinctions needed for a particular operational decision. This is related to the general study of sound abstraction in program analysis (P. Cousot and R. Cousot 1977).

Definition 4.1 (Decision-complete rendering)**.** *A rendering $r : \mathcal{S} \to R$ is* decision-complete *for admissibility D if there exists a function*

$$\overline{D} : r(\mathcal{S}) \times \mathcal{A} \longrightarrow \{\mathsf{true}, \mathsf{false}\} \quad \textit{such that} \quad D(s, a) = \overline{D}(r(s), a)$$

for every $s \in \mathcal{S}$ and $a \in \mathcal{A}$.

Proposition 4.2 (Rendering factorization criterion)**.** *A rendering r is decision-complete if and only if*

$$\forall s, t \in \mathcal{S},\ \forall a \in \mathcal{A}, \qquad r(s) = r(t) \implies D(s, a) = D(t, a).$$

Proof. If $D = \overline{D} \circ (r \times \mathrm{id}_{\mathcal{A}})$, then equal renderings give equal decisions by substitution. Conversely, suppose the displayed condition holds. Define $\overline{D}(v, a)$, for $v \in r(\mathcal{S})$, to be $D(s, a)$ for any s with $r(s) = v$. The condition ensures that this is independent of the chosen representative and therefore well-defined. By construction, $\overline{D}(r(s), a) = D(s, a)$ for all s, a. □

Proposition 4.3 (Indistinguishable states with divergent decisions)**.** *Suppose there exist $s, t \in \mathcal{S}$ and $a \in \mathcal{A}$ such that $r(s) = r(t)$ but $D(s, a) \neq D(t, a)$. Then no decision rule depending only on the pair $(r(s), a)$ can agree with authoritative admissibility at both s and t.*

Proof. Such a rule receives the same input in the two cases and so must return the same answer for both. Since the authoritative decisions differ, it is incorrect for at least one case. □

These propositions isolate a specific failure of *rendering-based authorization*. They do not prohibit a model from proposing useful actions based on partial information: the authoritative executor may still safely accept or reject each proposal using the full state. Moreover, a non-injective rendering can be decision-complete whenever all omitted distinctions are irrelevant to D.

4.2 Observation preservation is a separate question

Let $\eta : \mathcal{S} \to \Omega$ capture observations relevant to an application. A state transformation τ is observation-preserving on its domain when $\eta(\tau(s)) = \eta(s)$. This is not equivalent to invariant preservation or preservation of admissibility. For example, two states may have the same observation and different permissions. The correct relationship among rendering r, observation η, and decision D depends on the application. Any stronger claim about commuting representations requires explicit conditions on these maps and the transition language.

5 A Worked Quota-and-Permission Example

To make the roles of the invariant and guard nonredundant, let

$$\mathcal{S} = \mathbb{Z} \times \{\mathsf{reader}, \mathsf{writer}\}, \qquad \mathcal{A} = \{-1, 0, +1\},$$

and define

$$U((q,p),a) = (q+a,p), \qquad \mathcal{I}((q,p)) \iff 0 \le q \le 3.$$

The transition is defined even when the quota exceeds its permitted range. Hence the invariant is a genuine restriction imposed by the monitor, rather than a condition already built into U. Let the guard express a simple authorization rule:

$$G((q,p),a) = \mathsf{true} \quad \iff \quad (a = 0) \ \vee \ (p = \mathsf{writer}).$$

Thus a reader may propose a no-op but may not change the quota, even when the proposed successor satisfies the invariant.

Consider a coarse rendering

$$r((q,p)) = \min\{q, 2\},$$

which omits the permission field and distinguishes no quota values at or above 2. For illustration, all the following starting states satisfy the invariant:

Initial state	Action	Rendering	Decision	Committed quota
$(2, \mathsf{writer})$	$+1$	2	accept	3
$(3, \mathsf{writer})$	$+1$	2	reject	3
$(1, \mathsf{writer})$	$+1$	1	accept	2
$(1, \mathsf{reader})$	$+1$	1	reject	1

The first two rows share the same rendering and action but differ in quota admissibility. The last two share the same rendering and action but differ in guard authorization. In both pairs, Proposition 4.3 applies. Nonetheless, the monitor checks the authoritative state and gives the correct outcome in every row. This finite example is not an empirical language-model evaluation.

6 Relationship to Existing Methods and Companion Work

The separation between an untrusted component and a trusted monitor has clear precedents. Hoare-style reasoning supplies the underlying invariant discipline (Hoare 1969); runtime verification studies checking properties against executions (Leucker and Schallhart 2009); shielding places a protective decision layer between a learning agent and its environment (Alshiekh et al. 2018). Abstract interpretation provides a well-developed mathematical setting for reasoning about information lost under abstraction (P. Cousot and R. Cousot 1977). These works make it inappropriate to describe the monitor or rendering factorization as a wholly new mathematical mechanism.

More recent agent-focused research reinforces the importance of this comparison. *AgentSpec* develops a rule language and configurable runtime enforcement for LLM agents across multiple domains (Wang, Poskitt, and Sun 2026). *ShieldAgent* uses an agentic policy-reasoning approach with verifiable rules and evaluates it against agent-safety benchmarks (Chen, Kang, and Li 2025). Costa and collaborators develop *Fides*, a system based on information-flow control and confidentiality/integrity labels for restricting agent behavior (Costa et al. 2025). These systems pursue substantially broader policy or security objectives than the finite-state monitor studied here. The present work neither reproduces their evaluations nor claims to outperform them. Its narrower emphasis is an explicit authoritative state-and-commit semantics, a factorization condition on the model-visible state rendering, and a compact executable case study with its assumptions and measured limitations exposed.

The present paper fixes a minimal semantics for a *language-model proposal interface with persistent formal state*. It does not claim a novel synthesis algorithm for monitors, a new proof calculus, or empirical advantages over existing constrained or tool-using agent architectures. Retrieval-augmented generation (Lewis et al. 2020) changes what information a model can access but is not, by itself, a state-transition authorization mechanism.

The related manuscript *Proof-Carrying State Transitions for Reliable AI Systems* (R16) (Diamond 2026) studies an additional question: whether an untrusted proposer can attach a certificate to a suggested transition for separate verification. A direct invariant check in the present paper should not be confused with validating a portable proof certificate. The two mechanisms may be combined, but their soundness obligations are not interchangeable.

7 Executable Reference Monitor and Bounded Evaluation

7.1 Implementation boundary

We implemented a standard-library Python prototype realizing the quota-and-permission model of the preceding section. An immutable state record contains the integer quota and a reader or writer role; a typed action contains a delta from $\{-1, 0, +1\}$. The update function computes $(q + a, p)$ even when the quota would become invalid. A monitored executor serializes check and commit with an in-process lock, evaluates the guard on the authoritative state, checks the successor invariant, and records the result. Rejected actions leave the state unchanged. A strict JSON entry point parses exactly one integer delta and rejects unsupported or malformed proposals without modifying the state.

The prototype is an executable realization of the example semantics, not a formal verification of Python code or a hardened access-control boundary. In the experiments reported below, candidate actions are supplied by a deterministic test generator, not by a language model. An optional model-adapter implementation is described separately; no live-model results are included here. In particular, we distinguish evidence for a *monitored state transition* from evidence for an *LLM-driven application*.

7.2 Exhaustive single-step comparison

The test domain contains all eight invariant-satisfying initial states with $q \in \{0, 1, 2, 3\}$ and $p \in \{\mathsf{reader}, \mathsf{writer}\}$. For each such state we test every action in $\{-1, 0, +1\}$, resetting to the initial state for each case. The baseline applies U directly without consulting G or $\mathcal{I}$; the monitor implements M. The same 24 scripted proposals are presented to both.

Measured property (24 single-step cases)	Count
Admissible actions accepted by monitor	14
Inadmissible actions rejected by monitor	10
Invariant violations in monitored successors	0
Inadmissible successors under unguarded baseline	10
Out-of-range successors under unguarded baseline	4
Guard violations under unguarded baseline	8

The two baseline failure categories overlap: an unauthorized action can also move the quota out of range. Accordingly the union is ten, not the sum of four and eight. The unguarded baseline is deliberately minimal; its behavior should not be generalized to deployed agent frameworks that may apply their own checks.

7.3 Bounded traces, parser behavior, and state rendering

For each of the eight valid initial states, the harness enumerates all $3^5 = 243$ five-action sequences. This gives $1,944$ finite traces and $9,720$ monitored steps, with no observed violation of the quota invariant. The automated test suite also checks rejected malformed JSON, reader permission restrictions, out-of-range writer updates, a deterministic seeded proposal stress test, and linear audit-event ordering under concurrent calls within one Python process. All ten test methods passed in the recorded run.

The experiment also illustrates Proposition 4.3: the states $(2, \mathsf{writer})$ and $(3, \mathsf{writer})$ both render to 2 but give different decisions for action $+1$. A second collision arises between reader and writer at quota 1. These cases confirm the finite counterexamples for the chosen rendering, rather than establishing a new result about general language-model cognition.

Reproducibility. The accompanying archive contains a runnable monitor, a deterministic experiment harness, an automated test suite, JSON and CSV results, and a stand-alone HTML demonstration. The command `python3 code/experiment.py` regenerates the experiment files; `python3 -m unittest discover -s code` runs the tests. The prototype requires only the Python standard library. The HTML page illustrates the semantics interactively but is not the source of the reported measurements.

7.4 What remains untested

The mathematical theorem covers arbitrary finite proposal sequences under its assumptions; the experiments cover only explicitly specified finite cases. Passing tests cannot independently establish checker correctness, secure non-bypassability, or cross-process atomicity. Future work must run the connected model against the fixed harness, measure proposal quality and latency on meaningful tasks, compare with stronger guarded baselines, and evaluate whether the state schema captures real application requirements.

8 Connected-Model Prototype and Evaluation Protocol

8.1 Separating proposed actions from committed transitions

The accompanying software includes an optional model adapter that requests a structured action from an external language model. The reference interface accepts a natural-language instruction together with a deliberately lossy state rendering, and requests a single JSON object representing a quota change. The independent Python executor parses this response and evaluates the same guard and successor invariant used in the scripted experiments. The provider cannot write authoritative state through the model interface.

The connection supports an HTTPS language-model API and a locally running model service. The local browser interface communicates with a loopback-bound Python server; credentials, if required, remain server-side. The monitor compares an audit-log version captured before model inference with the version at proposed commit time, rejecting a response if another audit event has occurred during the intervening request, even if that event did not change state. This is a single-process research implementation, not a distributed-transaction guarantee or a tamper-resistant boundary.

8.2 Live evaluation design and evidentiary boundary

A fixed evaluation harness enumerates eight initial states and three natural-language instruction types (increase, decrease, no change). For each case, the experiment can record the raw model response, strict parsing success, agreement with the requested action, guard or invariant rejection, state after the decision, and model latency. This separates the model's ability to propose an intended action from the executor's enforcement of admissible state transitions. In particular, an incorrect proposal may still be safely rejected, whereas an authorized but unintended action is not ruled out merely by invariant preservation.

The model-connection code and browser round-trip have been exercised by automated tests with deterministic mock providers, including malformed output, unavailable providers, stale proposals, and invariant-boundary rejection. The combined software test suite contains 21 passing tests in the documented development run. These are interface and implementation tests, *not measurements of responses from an actual language model*. No model-dependent success rate, failure rate, latency distribution, or comparative advantage is claimed until a named provider and model are run under a recorded protocol with reproducible result files.

9 Scope, Limitations, and Open Questions

The included reference implementation has automated tests but no machine-checked correctness proof. It does not establish the correctness or security of a general machine-readable schema, parser, guard implementation, or audit log. Nor does the paper establish that formal invariants fully represent social, ethical, operational, or security requirements. Partial observability may reduce useful proposal quality even when it does not compromise monitor safety. A bypassable or non-atomic executor falls outside the model of Theorem 3.1.

The following questions define the next research steps:

1. Which transition languages are expressive enough for useful stateful agent tasks while retaining independently checkable guards?

2. When does a chosen state rendering factor admissibility, and when is incomplete information beneficial or harmful to the proposer?

3. What testable advantage arises from keeping state outside the model context rather than reconstructing state from a conversation?

4. Under what semantics can accepted steps carry portable certificates, as in R16 (Diamond 2026), without greatly expanding the trusted computing base?

5. What progress or responsiveness properties can coexist with strict monitoring and rejection of illegal proposals?

10 Conclusion

Formal-state augmentation separates the generation of candidate actions from the authority to commit persistent state changes. The monitoring semantics yields conditional invariant preservation for arbitrary finite proposal sequences. The rendering-factorization criterion identifies exactly when a model-visible state summary contains enough information to determine admissibility, while the worked example shows that omitted distinctions can matter even when the monitor itself remains safe. The architectural contribution is the explicit interface and resulting proof obligations, not a claim that elementary invariant reasoning is new. The accompanying reference monitor and scripted test suite provide finite, reproducible evidence of those semantics and their failure modes relative to an unguarded baseline. Claims about the behavior of a deployed language-model system require actual connected-model measurements, stronger implementation assurance, and measurements of proposal quality, decision accuracy, and operational cost.

References

Alshiekh, Mohammed et al. (2018). "Safe Reinforcement Learning via Shielding". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. DOI: 10.1609/aaai.v32i1.11797.

Chen, Zhaorun, Mintong Kang, and Bo Li (2025). "ShieldAgent: Shielding Agents via Verifiable Safety Policy Reasoning". In: *Proceedings of the 42nd International Conference on Machine Learning*. Vol. 267. Proceedings of Machine Learning Research, pp. 8313–8344. URL: https://proceedings.mlr.press/v267/chen25ae.html.

Costa, Manuel et al. (2025). *Securing AI Agents with Information-Flow Control*. DOI: 10.48550/arXiv.2505.23643. arXiv: 2505.23643 [cs.CR].

Cousot, Patrick and Radhia Cousot (1977). "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints". In: *Proceedings of the Fourth ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. ACM, pp. 238–252. DOI: 10.1145/512950.512973.

Diamond, Adrian (Oct. 2026). "Proof-Carrying State Transitions for Reliable AI Systems". AAD Systems Research manuscript, draft dated October 29, 2026 (R16).

Hoare, C. A. R. (1969). "An Axiomatic Basis for Computer Programming". In: *Communications of the ACM* 12.10, pp. 576–580. DOI: 10.1145/363235.363259.

Leucker, Martin and Christian Schallhart (2009). "A Brief Account of Runtime Verification". In: *The Journal of Logic and Algebraic Programming* 78.5, pp. 293–303. DOI: 10.1016/j.jlap.2008.08.004.

Lewis, Patrick et al. (2020). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". In: *Advances in Neural Information Processing Systems*. Vol. 33, pp. 9459–9474.

Vaswani, Ashish et al. (2017). "Attention Is All You Need". In: *Advances in Neural Information Processing Systems*. Vol. 30.

Wang, Haoyu, Christopher M. Poskitt, and Jun Sun (2026). "AgentSpec: Customizable Runtime Enforcement for Safe and Reliable LLM Agents". In: *Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE 2026)*, pp. 2938–2950. DOI: 10.1145/3744916.3764546.