

From Prompt to State: Verifiable Grounding and Operative Replay for LLM-Mediated Control

Tomas Laurenzo
Department of Critical Media Practices
University of Colorado Boulder
Boulder, Colorado, USA
tomas@laurenzo.net

Abstract—When a large language model controls software, its response does not act directly. The application must interpret the returned text as a command and execute that command through conventional program logic. Prompt logs record what was submitted to and returned by the model, while action logs record what the software subsequently did; neither necessarily explains the relation between the two. This paper develops an auditable account of that boundary.

We introduce a trace contract that connects an application-level model invocation, the returned response or a commitment to it, the command derived from that response, the applicable domain rules, and the resulting state and semantic events. Four separate verification predicates distinguish trace integrity, invocation reconstruction, grounding consistency, and operative replay. The contract is implemented in BatLLM, an open-source turn-based game in which local language models translate human instructions into a constrained command language.

Evaluation shows that all 1,080 recorded transitions replayed equivalently, production and independently implemented reference semantics agreed across 5,000 cases, and all tested semantic perturbations were rejected while equivalent serializations remained valid. The results demonstrate that reproducible generation is not required for verifiable execution: once a response has been recorded, its interpretation and computational consequences can be examined independently.

Keywords— *large language models, provenance, replay, grounding, verification, human–AI interaction*

I. INTRODUCTION

Consider an LLM-controlled game that records a human instruction, the model response C90, and a bot whose heading has changed by ninety degrees. The record looks complete and explains almost nothing. Which messages and which state were submitted to the model? Did the returned text actually ground to C90? Which rules interpreted that command? Do those rules entail the recorded state and events? Calling the model again answers none of this: inference is probabilistic, and its numerical behavior varies across execution environments [1].

The difficulty is that an LLM-mediated action looks like one operation and is in fact two. A model produces text under uncertain inference. An application then parses that text and applies conventional program semantics to it. Prompt-response logs describe the first regime. Action and state logs describe the second. Neither describes the seam between them, and the seam is where the interesting failures live: a plausible response that grounds to nothing, a valid command whose consequence nobody can reconstruct, a state change with no recorded warrant.

We study that seam through BatLLM [2], an open-source turn-based game in which two human players control software agents through natural-language instructions. A local model

returns a command; deterministic game code applies movement, rotation, shielding, shooting, collision, and damage rules. The command language is deliberately small, but the resulting transitions are non-trivial enough to separate linguistic evidence from executable consequence.

BatLLM was built as a teaching instrument. It exists so that students can watch their own language become an action and argue about the translation, and it will be used in a workshop course at the Instituto de Computación, Facultad de Ingeniería, Universidad de la República. This paper makes no claim about learning outcomes. It reports what the artifact had to become in order to be trustworthy in that role: a system in which the passage from prompt to state is recorded as evidence rather than asserted as behavior.

The paper defines a trace contract and four verification predicates that keep trace integrity, invocation reconstruction, response-to-command grounding, and operative replay apart. It implements that contract as a versioned research runtime with explicit text-retention profiles and an independently written reference semantics. It evaluates the implementation through replay, differential testing, semantic perturbation, benign-serialization controls, and storage and verification cost. The relation under verification is specific: given a recorded model response, the paper establishes how BatLLM grounded that text and whether the resulting command produced the recorded state. It does not ask a later invocation to reproduce the response.

II. RELATED WORK

Games and interactive environments provide structured settings for observing LLM agents. AgentBoard measures progress over multi-turn tasks [3], while Cradle studies general computer control through screen-based interaction [4]. BatLLM differs in purpose: the game is used here not to measure how well a model plays, but as an inspectable domain in which the transition from language to executable state can be verified.

Provenance standards supply the broader vocabulary. W3C PROV models entities, activities, agents, and derivations [5]. OpenTelemetry's generative-AI conventions define attributes for ordered model inputs and outputs and warn that captured message content may be sensitive [6]. PROV-AGENT extends provenance to prompts, responses, decisions, and workflow lineage [7], and a recent survey organizes evidence tracing and execution provenance across observability, tools, memory, audit, and recovery [8].

Record-and-replay work addresses adjacent objectives. AgentRR reuses successful trajectories as later experience [9]. Prompt Scene Investigation captures prompt-response evidence and re-enacts model behavior under controlled conditions [10]; agrepl replays captured external interactions without outbound access [11]. Paduraru et al. combine

message-action traces, executable contracts, replay, and fault injection for agent orchestration [12]. Their assurance framework is the closest systems precedent, but it addresses orchestration-wide behavior and external side effects. BatLLM instead makes the application-level relation among invocation, returned language, grounded command, and deterministic consequence the object of independent verification.

III. A CONTRACT FOR THE LANGUAGE-TO-STATE BOUNDARY

A. Execution model

For step i , let u_i be the human instruction, h_i the ordered conversation history, s_i the pre-action state, θ_i the declared model configuration, and ρ_i the frozen domain rules. The application constructs the invocation record visible at its provider-adaptor boundary,

$$a_i = A(u_i, h_i, s_i, \theta_i)$$

and receives response text y_i from provider M . A parser grounds the text into command c_i ; a transition function produces the next state and ordered semantic events:

$$c_i = P(y_i), \quad (s_{i+1}, e_i) = \delta(s_i, c_i, \rho_i)$$

The trace unit is

$$T_i = \langle u_i, h_i, s_i, a_i, \theta_i, y_i, c_i, \rho_i, e_i, s_{i+1} \rangle$$

The adaptor record contains the model, ordered messages, options, and stream setting passed by BatLLM, together with the declared provider and endpoint. It is not a capture of provider-library internals, HTTP bytes, hidden defaults, model weights, or serving binaries.

B. Verification predicates

The contract separates four questions that a single notion of reproducibility keeps fused.

V1 — Trace validity. Are the schema, identifiers, sequence, state continuity, final states, commitments, and ordered play chain internally consistent?

V2 — Invocation evidence. Does the retained evidence support the invocation attributed to the application? With full text, the verifier rebuilds the current user message and history from separately recorded prompt, state, system instructions, and context policy, then compares them with a_i .

V3 — Grounding consistency. When response text is retained, does re-applying P produce the recorded command and validity status? Provider failures are represented separately and resolve operatively to ERR.

V4 — Operative replay. Does re-evaluating $\delta(s_i, c_i, \rho_i)$ reproduce both s_{i+1} and e_i ? BatLLM compares discrete state exactly, and position and rotation with absolute tolerance 10^{-6} .

Table I makes the disclosure trade-off explicit. Full retention supports all four checks. Redacted traces keep request structure and commitments; hash-only traces keep commitments alone. Both preserve operative replay, and neither can establish independently which command the hidden response would produce. Hashes here are consistency commitments, not confidentiality or authentication: low-entropy text can be guessed, and a party able to replace a

complete trace and every external copy can recompute all internal values.

TABLE I. EVIDENCE AVAILABLE BY TEXT-RETENTION PROFILE

Predicate	Full	Redacted	Hash-only
V1 trace	yes	yes	yes
V2 invocation	reconstruct	structure + hash	hash
V3 grounding	verify	no	no
V4 replay	verify	verify	verify

A successful verification therefore has a precise meaning. Assuming deterministic P and δ and the semantics identified by the trace, V3 and V4 establish that the recorded command equals $P(y_i)$ and that the recorded state and events equal $\delta(s_i, P(y_i); \rho_i)$. Historical authenticity additionally requires a trusted recorder or an independently preserved signature, timestamp, or commitment. Verification does not imply that another invocation of M would return y_i .

IV. IMPLEMENTATION

BatLLM normalizes model text into movement (M, or M followed by a distance), clockwise and counter-clockwise rotation (C or A followed by an angle), shooting (B), and shield operations (S, S1, S0); unsupported text resolves to ERR. Gameplay is implemented as a pure transition module independent of the Kivy interface. The replay projection includes each bot's health, position, rotation, and shield state. Prompt and diagnostic fields remain invocation evidence rather than gameplay state.

The schema-v3 research runtime is a headless path that uses the production parser and transition engine. Immediately before an adaptor call it materializes the ordered messages, declared provider and endpoint, requested model, generation options, and stream setting. It then records the response or error, the normalized command, the frozen rules, pre- and post-state, ordered semantic events, timestamps, latency, and attempt count. If every attempt fails, the pending user message is removed from subsequent conversation history, matching BatLLM's rollback policy. The reference evaluation uses a scripted provider to isolate trace semantics from model and network variability; a Ollama adaptor supplies a live local-model integration path.

Canonical JSON uses UTF-8, lexicographically sorted keys, compact separators, and finite numbers. Separate commitments cover protected text, retained requests, transitions, play records, the ordered play chain, and the session envelope. The verifier reports predicate-specific failures and exits non-zero. So that replay is not evaluated by invoking production code twice, the artifact also contains a reference semantics that imports no production gameplay module and separately implements command parsing, movement bounds, rotation, shields, projectile traversal, collision, damage, state normalization, and semantic events. This is differential evidence, not proof: both implementations can still share a specification error. Source, schema, experiments, committed results, and build instructions are available under research/urucon2026 in the BatLLM repository [2].

V. EVALUATION

A. Protocol

The evaluation asks whether valid traces satisfy the evidence available under each retention profile; whether production and reference semantics agree; whether the verifier distinguishes semantic inconsistency from harmless representation change; and what storage and verification cost the trace representation imposes.

The corpus contains 60 sessions and 1,080 transitions: twenty sessions per retention profile, eighteen transitions per session. It varies shared and independent histories, augmented and unaugmented prompts, game parameters, every documented command form, and invalid text. Responses are scripted. The design tests the trace and transition contract, not model competence, provider fidelity, or pedagogical effect. All 60 traces validate against the published JSON Schema Draft 2020-12 with date-time format checking.

Differential testing uses a fixed seed and 5,000 targeted and random cases spanning positions, rotation, health, shields, rules, commands, invalid text, and projectile outcomes, comparing command normalization, resulting state, ordered events, and projectile path between the production and reference implementations.

The perturbation experiment alters eleven classes of evidence: command, retained response, retained invocation, human prompt, state supplied to the model, rules, pre-state, post-state, events, deletion, and reordering. Applicable play-local mutations are placed early, midway, and late in each session; rule changes are applied at round level. Before verification, every commitment derivable from mutated retained content is recomputed, so detection cannot rest on stale hashes. A control reserializes each valid trace as compact ASCII, pretty Unicode, and recursively reversed key order. Storage is measured on canonical UTF-8 JSON and gzip level 9; verification timing is measured in memory over the same corpus.

TABLE I. REFERENCE ARTIFACT RESULTS

Measure	Result
Schema-valid / semantically valid sessions	60 / 60
State / event replays	1,080 / 1,080
Full invocation reconstructions	360
Retained-text groundings verified	360
Differential normaliz. / event / path cases	5,000 each
Re-anchored perturbations rejected	1,560 / 1,560
Benign serializations accepted	180 / 180
Raw bytes/play: full / redacted / hash	4,374 / 4,741 / 2,203
Mean gzip bytes per play	439
Median in-memory verification	29.45 ms/session; 611 plays/s

B. Results

Every session satisfied the schema and the semantic checks available under its retention profile (Table II). All 1,080 recorded states and event lists replayed equivalently. Full retention reconstructed 360 adapter-boundary invocation records and verified 360 response-to-command mappings; the 720 reduced-retention transitions remained replayable without being reported as independently grounded.

Production and reference semantics agreed in all 5,000 cases for normalization, state, events, and projectile paths. The verifier rejected all 1,560 applicable re-anchored perturbations and accepted all 180 benign variants. These are coverage results for the declared fixtures, not estimates of detection probability in deployment.

Raw storage was 4,374, 4,741, and 2,203 bytes per play for full, redacted, and hash-only retention; compression brought the overall mean to 439 bytes per play. Redacted requests are larger than full requests in this corpus because each message body is replaced by a structured record carrying a digest and type metadata — the redaction costs more bytes than the text it hides. Median in-memory verification was 29.45 ms per session, or 611 plays per second on the recorded test environment.

VI. DISCUSSION

The contract turns one vague reproducibility question into three answerable audit questions: what did the application submit, how was the returned language grounded, and did the grounded action entail the recorded consequence? The separation matters because a replayable command log is not evidence of what a model said, and a prompt-response log is not evidence of what the software then did. Full traces bind both. Reduced-retention traces make explicit which part of that relation is unavailable.

The design transfers to systems in which model text crosses a visible grounding boundary into a command language and the subsequent semantics can be frozen or otherwise reconstructed. Tool-calling interfaces, workflow languages, simulations, and constrained robotic controllers may meet that condition, but each domain needs its own state projection, event vocabulary, and independent semantic tests. BatLLM demonstrates the pattern in one domain; it does not establish cross-domain generality.

The distinction between semantic consistency and provenance remains essential. Re-anchored perturbations show that the verifier checks relationships among fields rather than stale hashes alone, and benign serializations show that acceptance is semantic rather than byte-for-byte. A comprehensive rewrite can nonetheless remain internally valid if every external anchor is replaced with it. External chain of custody is therefore complementary to operative replay, not supplied by it.

VII. CONCLUSION

The evaluation uses scripted responses and a deterministic game domain. It does not measure live-provider completeness, model quality, human usefulness, or learning. Systems with stochastic post-grounding behavior must additionally record random state or external effects. The separate reference semantics reduces circular testing but cannot exclude a shared specification error. Adapter records do not capture transport bytes and do not authenticate model weights or serving code. Reduced retention prevents independent grounding verification, and the graphical application still exports schema v2; the claims here concern the explicit v3 research runtime.

The central result is that reproducible generation is not a precondition for verifiable execution. Once a response has been recorded, its grounding and its consequences can be

checked independently under explicit rules. By preserving evidence at the application boundary and replaying the domain transition, BatLLM gives a concrete account of how a human instruction became computational state — and, for the students who will use it, an account they can open, contest, and re-run.

ACKNOWLEDGMENT

BatLLM was supported by the Center for Humanities & the Arts and the Research & Innovation Office at the University of Colorado Boulder.

REFERENCES

- [1] J. Yuan et al., "Understanding and mitigating numerical sources of nondeterminism in LLM inference," in *Adv. Neural Inf. Process. Syst.*, vol. 38, 2025, pp. 169819–169851.
- [2] T. Laurenzo, BatLLM, ver. 0.3.6, computer software, GitHub repository krahd/BatLLM, Jul. 2026.
- [3] C. Ma et al., "AgentBoard: An analytical evaluation board of multi-turn LLM agents," in *Adv. Neural Inf. Process. Syst.*, vol. 37, Datasets and Benchmarks Track, 2024, pp. 74325–74362, doi: 10.52202/079017-2365.
- [4] W. Tan et al., "Cradle: Empowering foundation agents towards general computer control," in *Proc. 42nd Int. Conf. Mach. Learn. (ICML)*, PMLR, vol. 267, 2025, pp. 58658–58725.
- [5] T. Lebo, S. Sahoo, and D. McGuinness, Eds., "PROV-O: The PROV ontology," W3C Recommendation, Apr. 2013.
- [6] OpenTelemetry Authors, "Semantic conventions for generative AI," OpenTelemetry Semantic Conventions, ver. 1.37.0, 2025.
- [7] R. Souza et al., "PROV-AGENT: Unified provenance for tracking AI agent interactions in agentic workflows," in *Proc. IEEE 21st Int. Conf. e-Science*, 2025, pp. 467–473, doi: 10.1109/eScience65000.2025.00093.
- [8] Y. Wang et al., "From agent traces to trust: A survey of evidence tracing and execution provenance in LLM agents," arXiv:2606.04990, 2026.
- [9] E. Feng et al., "Get experience from practice: LLM agents with record & replay," arXiv:2505.17716, 2025.
- [10] L. F. Ledjaki, Y.-C. Chang, M. Loutis, and E. Aïmeur, "Prompt scene investigation: Uncovering the evidence in LLMs," in *Companion Proc. ACM Web Conf.*, 2026, pp. 284–290, doi: 10.1145/3774905.3795469.
- [11] R. Mudasiru, "Deterministic replay for AI agent systems," arXiv:2607.16200, 2026.
- [12] C. Paduraru, P.-L. Bouruc, and A. Stefanescu, "A trace-based assurance framework for agentic AI orchestration: Contracts, testing, and governance," in *Proc. 21st Int. Conf. Eval. Novel Approaches Softw. Eng. (ENASE)*, 2026, pp. 420–427, doi: 10.5220/0014840300004015.