

TRACE-RPG: A Trace-Linked Symbolic Commit Gate for Generated Events in an Interactive Game World

Anonymous Author(s)

Abstract—Large language models can generate expressive quests and dialogue, but fluent or schema-valid text does not establish that a proposed event is executable, authorized, or consistent with canonical game state. We present TRACE-RPG, a symbolic commit gate that treats every generated event as an untrusted transaction proposal. A type-strict parser, an externally supplied action policy, and seven deterministic checks grouped into six state-relative families decide each admitted transition; a failing candidate may enter bounded repair for at most $K + 1$ recorded attempts, and a passing candidate is revalidated before mutation. The counterexample-guided operator ρ reads only the prior candidate and its typed validation errors, never authoritative state. Unkeyed SHA-256 content links, state-semantic replay, and assignment-complete accounting make every terminal outcome auditable. Three evidence lanes stay separate: a deterministic offline conformance pilot over one authored world, a single-model live screening extension, and a closed-world knowledge-graph link-proposal simulation with no runtime authority. The offline lane establishes mechanism conformance only; at $K=1$ the live lane observed 5/5 guided versus 0/5 blind commits solely in a deliberately repairable policy-blind regime, so it remains pilot-only. No lane establishes population efficacy, model superiority, player experience, or commercial-engine performance.

Index Terms—game artificial intelligence, interactive narrative, symbolic validation, constrained generation, runtime enforcement, reproducibility

I. INTRODUCTION

INTERACTIVE generation is more than a language task. A candidate event may be natural yet require an absent item, assert a fact unknown to its speaker, reveal forbidden quest information, introduce an unauthorized effect, or regress a quest stage. Executable text environments make the distinction between a plausible utterance and a valid transition explicit [1], [2]. Grounded dialogue and personalized quest systems further show the value of quest, entity, and graph context [3], [4], but context does not itself authorize canonical mutation. TRACE-RPG treats generated game events as untrusted transaction proposals and permits canonical state mutation only through an external, deterministic commit gate. Fig. 1 reads one such transaction left to right.

Structural constraints solve an adjacent problem: grammar-constrained decoding, query languages, and incremental parsers restrict generated structures [5], [6], [7], yet a syntactically admissible event can still be false relative to the current world. Retrieval and memory supply useful evidence [8], [9],

TABLE I
CONTRIBUTIONS, PRIOR TOPICS, EVIDENCE LANES, AND INFERENCE CEILINGS (GENERATED FROM THE STRUCTURED MATRICES)

ID	Contribution	Prior topics (refs)	Lanes	Inference ceiling
C1	Trust-boundary contracts	T1, T2, T3 (15)	E1, ENG1	encoded-field contract conformance
C2	Validate–repair–commit controller	T1, T8 (14)	E1, ENG1	mechanism behavior on frozen cases
C3	Audit-linked evidence layer	T4, T7, T8 (6)	E1, ENG1	integrity/replay on named fixtures
C4	Assignment-complete harness	T4, T6, T7 (18)	E1	exact designed-case accounting
C5	Counterexample-guided operator ρ	T1, T8, T9 (8)	E1, E2	frozen classes + one regime screen

T1 = grounded game worlds; T2 = retrieval/memory agents; T3 = structured generation; T4 = environments/benchmarks; T5 = player experience; T6 = human/LLM evaluation; T7 = statistics/reproducibility; T8 = planning/runtime interposition; T9 = feedback repair. Lanes are defined in Table IV. All rows are authored-fixture verified; none supports an efficacy claim.

[10], but retrieved content is not a state-transition authority either.

This paper presents TRACE-RPG, an implementation that combines these concerns as an auditable transaction protocol. Its current contributions are:

- 1) **C1—Contracts:** versioned trust-boundary contracts and a type-strict parser that rejects unknown keys and ambiguous types at proposal and replay boundaries (Sec. IV; E1);
- 2) **C2—Controller:** a validate–repair–commit controller with seven checks across six state-relative families, bounded repair, unchanged fallback, and pre-commit revalidation (Sec. IV-B; E1, ENG1);
- 3) **C3—Audit:** SHA-256-linked records, state-semantic replay, and episode-continuity checks (Sec. IV-C; E1, ENG1);
- 4) **C4—Harness:** an assignment-complete offline conformance harness that keeps a classified terminal row even when no proposal trace can exist (Sec. V; E1); and
- 5) **C5—Repair:** the counterexample-guided operator $\rho(a, E)$, compared with blind retry and a state-reading oracle across frozen repairability classes (Sec. IV-B; E1, E2).

Table I, generated from the machine-checked contribution matrix, binds C1–C5 to their prior-work topics, evidence lanes, and inference ceilings; Table IV defines the lanes.

The lanes are non-interchangeable (Table IV): **E1** is a deterministic conformance pilot over one authored world and

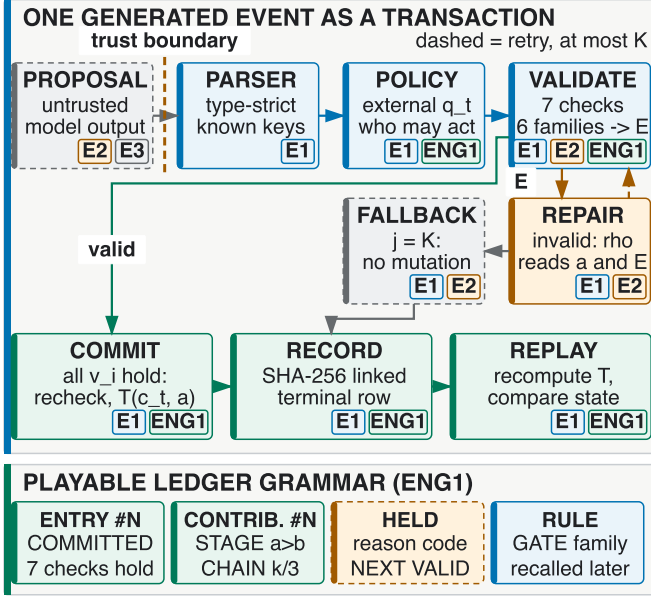


Fig. 1. One generated event as a transaction. A typed candidate crosses the trust boundary; seven checks in six state-relative families consult external policy and c_t . Typed errors E feed bounded ρ (dashed) or unchanged fallback; passing candidates are rechecked, committed, recorded, and replayed. Tags mark measurement scope, not claims; the lower strip is the committed-snapshot ledger grammar.

measures encoded mechanism behavior, not population efficacy; **E2** is a five-call-per-cell live screening capped at pilot-only by its unpinned revision, uncontrolled seed labels, and constructed states; **E3** scores authored typed-link proposals in a closed world without runtime retrieval; and ENG1 embeds the same gate in a Godot 4.7.1 slice, where the player reads holds as rules and commits as contributions (Fig. 3). There are no participants, affect experiments, or live Python–Godot round trips.

II. RELATED WORK AND RESEARCH GAP

A. Interactive Narrative and Game Agents

KNUDGE grounds branching NPC dialogue in quest and entity specifications [3]; knowledge-graph quest generation [4], memory–reflection–planning agents [11], and trainable role-playing agents [12] address complementary aspects of grounded, believable behavior. Player-facing studies evaluate preference and perceived dialogue quality [13], [14], [15], and a 2025 preprint reports role-sensitive stability–improvisation trade-offs [16]; both motivate separating hard admissibility from optional naturalness and player-experience constructs.

Experience-driven generation [17], observer-labelled engagement [18], and limits of vision-language engagement inference [19] motivate a future adaptation track; affect is non-authoritative and absent from the present evidence.

B. Environments and Benchmarks

TextWorld and ScienceWorld expose explicit interactive state [1], [2], MineDojo adds open-ended embodied tasks [20], and GVGAI separates agent, game, and content tracks [21].

AgentBench, AgentBoard, SOTOPIA, and BALROG evaluate agents across heterogeneous, process-diagnosed, social, and long-horizon game settings [22], [23], [24], [25]; automated play-style agents complement rather than replace human evaluation [26]. None prescribes a transactional authorization boundary.

C. Constraints, Neuro-Symbolic Validation, and Retrieval

Constrained decoders improve output structure [5], [6], [7]; TRACE-RPG instead treats structure as an input condition to independent state-relative authorization. The same generate-and-verify division is established outside games: LLM-Modulo frameworks let a model propose plans that external sound critics accept or reject [27], and SayCan admits a language-proposed skill only after a state-dependent affordance check [28]. AgentSpec expresses runtime rules for LLM agents as trigger–predicate–enforcement triples [29]; TRACE-RPG fixes the predicates to game state and adds a repair contract and replayable records. Setting the DC grounds D&D agents in tool calls with precondition checks [30], whereas TRACE-RPG admits free-text proposals only after parsing and external policy evaluation. STORY2GAME lets the LLM generate the action preconditions and effects it plays under [31]; here the policy is authored independently of the proposer. TRACE-RPG instantiates the critic as a deterministic, state-relative commit gate that returns typed counterexamples. Game-specific comparators span complementary boundaries: PANGeA validates content by asking an LLM to judge and refine it against designer rules [32]; IVIE incrementally generates interactive-fiction worlds with symbolic validation [33]; and a 2026 successor restricts the LLM to selecting authored world-state transformations [34], gaining consistency by construction while leaving this paper’s transaction, audit-record, and repair-operator questions as separate evidence targets. No comparison implies superiority: validity is only validity for the predicates a system encodes.

G-Retriever, HippoRAG, and Mem0 address subgraph retrieval and long-term memory [8], [9], [10]; Generative Agents and Character-LLM address behavioral memory and persona [11], [12]. TRACE-RPG may consume such context but denies it commit authority; their benefits are not measured here.

D. Auditability and Reproducibility

Process metrics motivate retaining intermediate failures [23]; robust evaluation cautions against overinterpreting sparse stochastic runs [35]. Reproducibility and environmental reporting motivate frozen artifacts and explicit measurement boundaries [36], [37]. Human and LLM evaluation require construct, rater, and bias controls [38], [39], [40], [41], while future non-inferiority and participant analyses require justified margins and random-effects structures [42], [43]. Record-and-replay has been proposed as an agent paradigm for experience reuse [44]; TRACE-RPG claims only semantic replay of admitted state transitions.

The bounded novelty is the implemented combination, not the invention of its individual parts. Prior work separately demonstrates runtime enforcement for general agents [29],

symbolic state management in generated game worlds [30], [31], feedback-driven repair [45], and agent record-and-replay [44]; TRACE-RPG studies their intersection at a game-state commit boundary, where parsed proposals are checked against an independent policy and deterministic predicates, optionally repaired from bounded validator feedback, and recorded for semantic replay. Action interposition is long established: narrative mediation intercepts an action before it executes [46]; classical planning defines applicability through preconditions and effects [47]; runtime enforcement admits only permitted transitions [48]; transaction processing leaves the prior state intact on abort [49]; authored-logic interactive drama authors the social practices that afford a character’s utterances [50]; and belief-aware narrative planning reasons over what characters know, which our knowledge and disclosure predicates encode far more coarsely [51]. C1–C2 place strict contracts and a validate–repair–commit controller over this lineage; C3–C4 add checksum-linked records, state-semantic replay, episode continuity, and assignment-complete accounting that retains cases with no proposal trace; C5 isolates a repair operator that consumes only typed pre-execution validator errors. We do not infer that a feature absent from a report is absent from every implementation.

III. PROBLEM DEFINITION AND GUARANTEE BOUNDARY

For exposition, the controller input used by validation at step t is decomposed as $c_t = (G_t, q_t)$, where G_t denotes typed snapshot fields and q_t denotes the externally authored action-policy, disclosure, and quest-stage constraints stored with that snapshot. Terminal record history $h_{<t}$ is maintained separately from the `WorldState` snapshot; no validation predicate reads it, and the implementation does not reconstruct state from the log. Retrieval, narrative scores, model rationales, memory summaries, and any future affect estimate are non-authoritative proposal context.

An untrusted proposer emits candidate a_t . Strict parsing first enforces the shared candidate-key contract, including rejection of unknown top-level fields, and establishes the declared schema. The validator returns

$$V(c_t, a_t) = (v_{\text{policy}}, v_{\text{pre}}, v_{\text{reach}}, v_{\text{know}}, v_{\text{disc}}, v_{\text{quest}}, E_t), \quad (1)$$

where E_t is a structured error set. Mutation is

$$c_{t+1} = \begin{cases} T(c_t, a_t), & \bigwedge_i v_i = 1, \\ c_t, & \text{otherwise.} \end{cases} \quad (2)$$

The controller asserts the following implementation invariants under its encoded contracts; the pilot tests their mechanisms with deterministic valid, invalid, repair, fallback, replay, and fault-injection fixtures rather than presenting general theorems. **I1 (exhausted-failure immutability)**: if no returned candidate through budget K validates, or proposal/controller execution fails before commit, the returned state equals the supplied prior state. **I2 (bounded recorded attempts)**: provided every proposal and repair callback returns, repair budget $K \geq 0$ admits at most $K + 1$ recorded candidate attempts; a successful commit adds one defensive validation immediately before mutation. The runtime does not enforce callback

TABLE II
ENCODED VALIDATOR BOUNDARY

Predicate	Rejects	On failure
Action policy	unknown action/type	unchanged
Precondition	absent/false requirement	unchanged
Reachability	inaccessible object	unchanged
NPC knowledge	undeclared known fact	unchanged
Disclosure	forbidden fact	unchanged
Quest	ineligible/regressive stage	unchanged

deadlines, so I2 is not a wall-clock termination guarantee. **I3 (effect authorization)**: an accepted candidate contains no declared fact effect in `CandidateAction.effects` outside `ActionPolicy.allowed_effects`, and any quest-stage target is explicitly authorized by that policy. **I4 (state-semantic replay)**: with the frozen schemas and deterministic transitions, an accepted trace reconstructs its stated terminal state.

These invariants assume correct policy authorship, faithful extraction into declared candidate fields, deterministic software versions, returning callbacks, and single-process file ownership. Replay revalidates the recorded structured candidate and transition; it does *not* verify the provenance, prompts, or stochastic generation of intermediate repairs, and natural-language implications omitted from structured fields remain outside the guarantee.

IV. TRACE-RPG ARCHITECTURE AND ALGORITHMS

A. Trust Boundary and Contracts

Fig. 1 places the model or recorded adapter outside the canonical owner. The parser rejects ambiguous numeric and collection types for known fields and unknown top-level candidate fields; the action policy is supplied separately and defines required preconditions and allowed effects; the versioned game bridge carries observations, proposals, validations, rejections, fallbacks, or commits, and only a valid commit authorizes canonical mutation. This decouples interfaces without demonstrating commercial-engine portability. Table II groups the seven implementation checks into six semantic families: quest eligibility and quest-stage monotonicity remain separate checks. This encoded boundary is not a claim of semantic completeness.

B. Validate–Repair–Commit

Fig. 2 gives the controller’s dependency-free pseudocode. “Record” denotes an append to the in-memory trace representation; persistent writing occurs only after terminal consistency checks.

Validation errors are exposed to a repair callback as structured error codes. Repair never directly mutates state, and exhaustion returns an unchanged deterministic fallback. Fig. 1 shows the terminal paths. The pilot compares two implemented callbacks. The *counterexample-guided operator* consumes only the prior candidate a and the validator’s structured error set E , never the authoritative state:

$$\rho(a, E) = a \oplus \bigcup_{e \in E} \delta(e), \quad (3)$$

```

1:  $a \leftarrow \text{PROPOSE}(c); j \leftarrow 0$ 
2: repeat
3:    $(ok, E) \leftarrow V(c, a); \text{RECORD}(a, E)$ 
4:   if  $ok$  then
5:     assert  $V(c, a).ok$ ; return  $\text{COMMIT}(c, a)$ 
6:   if  $j = K$  then return  $\text{FALLBACK}(c)$ 
7:    $a \leftarrow \text{REPAIR}(c, a, E); j \leftarrow j + 1$ 
8: until terminal

```

Fig. 2. Validate–repair–commit procedure. At most $K + 1$ candidate attempts are recorded; the successful path performs a defensive validation before mutation.

TABLE III
PER-ERROR-CLASS EDITS OF THE GUIDED OPERATOR ρ

Validator error class	Edit $\delta(e)$ on the candidate
Policy precondition omission	insert the named predicate
Policy effect omission	insert the named effect
Policy effect violation	drop the named effect
Unsatisfied precondition	drop the named predicate
Unauthorized stage mutation	clear <code>quest_stage_effect</code>
Unknown action type	no-op (no registered alternative)
Unreachable object; knowledge, disclosure, and stage classes	no-op (candidate-local edit would launder or needs state)

where each $\delta(e)$ is the deterministic per-error-class edit of Table III and $\oplus$ applies the resulting field edits. Each edit touches at most one candidate field per error entity, so the number of changed fields per attempt is bounded by $|E|$; applying ρ twice with the same error set is a fixed point; and an entity simultaneously demanded and rejected on the same field is declared irreparable and left untouched. The *state-reading reference callback* instead discards the error set and reconstructs the policy-governed fields (preconditions, effects, and both quest-stage fields) directly from the authoritative state; it deliberately never edits declaration-integrity fields (required objects, used facts, disclosed facts), because undeclaring a dependency or disclosure would launder the violation past the gate rather than repair the candidate. It is an oracle upper bound over state-readable repairs, and its outcomes must not be read as evidence about any deployable repair method.

C. Integrity, Semantic Replay, and Accounting

Each result row carries an unkeyed SHA-256 *integrity checksum*, and proposal outcomes link to a trace checksum; these detect accidental or tested content changes but are not signatures, writer identity, or protection against an actor who rewrites content and checksums. Semantic replay strictly parses the prior state and every recorded candidate/validation pair, requires every nonfinal attempt to be invalid, recomputes the terminal transition, compares the stated final state, and requires continuity between adjacent JSONL records; it does not re-execute the repair step that produced candidate $j + 1$.

The terminal record classes are commit, symbolic fallback, adapter failure, and controller failure. A controller-failure row remains in the frozen assignment denominator and leaves state unchanged, but it may legitimately lack a completed proposal trace because no candidate outcome was produced. Accordingly, result completeness does not imply

TABLE IV
EVIDENCE LANES AT A GLANCE: DESIGN, UNIT, COMPARISON, AND CEILING (GENERATED FROM THE EXPERIMENT MATRIX)

Lane	Design	Unit / budget	Comparison	Ceiling
E1	Offline conformance: authored frozen fixtures, one world	13 gate; 12 repair/arm; 10 faults; 7 adapter; 3 guards	reject vs blind vs ρ vs oracle	mechanism conformance
E2	Live RQ2 screening: one hosted proposer, matched candidate	5 cells $\times$ 5 calls; $K=1$	ρ vs blind per regime	pilot-only transfer screen
E3	KG/ontology simulation: closed-world typed-link holdout	43 nodes; 106 ref. edges; 24 typed; 210 scores	degree baseline vs 6 fixed strategies	construction result only
ENG1	Godot/Web engineering: engine-local policy mirror, no participants	4 fixtures; 52 checks; 8 smoke; 5 archetypes	authored fixture paths	engine-local conformance

Lanes are non-interchangeable with different denominators; none establishes population efficacy, model ranking, player experience, retrieval benefit, or commercial-engine performance.

trace existence for every terminal row. The pre-execution assignment key contains run, arm, scenario, seed, model, revision, controller-configuration hash, assignment-input hash, and prior-state hash. Aggregation rejects duplicate, missing, or unexpected keys and separately reports symbolic, adapter, and controller failures.

V. EVIDENCE LANES AND BOUNDARIES

Table IV fixes each lane’s design, unit, comparison, and ceiling before any count is read.

A. E1 Offline Conformance Design

The pilot asks whether (PQ1) valid candidates commit while named invalid-policy cases remain unchanged; (PQ2) rejection-only, blind unchanged retry, ρ , and the state-reading reference callback produce the expected outcome in every frozen repairability class; (PQ3) frozen checksum, state-semantic, linkage, and continuity mutations are rejected by their designated check operations and an injected second-file append failure triggers the pair-write rollback check; (PQ4) strict adapters and assignment manifests retain exactly one classified terminal row per assigned case or fail closed; and (PQ5) proposal and replay parsers both reject an otherwise-valid candidate carrying one unknown top-level key.

Cases are purposively designed conformance fixtures in four domains (validator/state isolation, repair-arm comparison, record/trace fault injection, adapter/accounting contracts), not samples from a model or game population. Every repair fixture declares a frozen repairability class before execution: *guided-repairable* (the error payload alone suffices for ρ), *oracle-only* (repair needs the authoritative state), or *irreparable* (no arm may commit); each class’s expected outcome per arm is part of the fixture contract, unique cases form the denominators, and all expectations precede execution.

Measures are exact counts: validator-class agreement; rejected or failed cases with state mutation; commit and attempt outcomes by arm and repairability class; detectable faults rejected by their designated operation; replayed traces; rejected manifest violations; and proposal/replay parity on

TABLE V
EXACT REPAIR-ARM COUNTS BY REPAIRABILITY CLASS (DESIGNED
FIXTURES)

Arm	Info source	Commits / cases			Mean edits ^e Fields
		G-rep. ^f	O-only ^f	Irrep. ^f	
Rejection-only ($K=0$)	none	0/5	0/1	0/6	—
Blind unchanged retry	none	0/5	0/1	0/6	—
Counterexample- guided ρ	error set E	5/5	0/1	0/6	1.0
State-reading oracle	state+policy	5/5	1/1	0/6	1.0

^eMean changed candidate fields over committed repairs (minimality proxy); failed arm executions left the state unchanged. ^fCommits/cases. G-rep. = guided-repairable, O-only = oracle-only, Irrep. = irreparable (frozen repairability classes).

the unknown-key regression, which is parser-contract parity rather than a safety outcome. Exception class alone does not identify a stable detector layer, so no detector attribution is reported, and no p -values or population intervals are attached. Adapter cases carry JSON-Schema-pinned synthetic latency and token constants that exercise accounting propagation only. The frozen offline packet binds fixtures, assignments, schemas, results with embedded traces, code revision, and checksums, following reproducibility and measurement-boundary guidance [36], [37].

B. E2 Live RQ2 Screening

The live extension compares blind unchanged retry with ρ at matched $K=1$: per cell, five seed-indexed calls to the hosted proposer `gpt-5.6-sol` request one candidate, and the same candidate feeds both arms, eliminating within-call proposal differences. Conditions vary policy visibility and whether a trivial zero-effect route exists. The frozen packet retains the current cells and the superseded `v1` diagnostic, binds every receipt by SHA-256, and is labelled pilot-only. Because seed labels do not control the hosted sampler, the revision is unpinned, token accounting is unavailable, and constructed states are not population samples, we report exact counts and failure classes without p -values, confidence intervals, or model ranking.

C. E3 KG/Ontology Link-Proposal Simulation

The third lane is an engineering-only closed-world simulation, separate from the authoritative `WorldState`: 43 graph nodes and 106 reference edges, 24 reviewed typed edges, passing type/domain/range checks and six competency questions, with six authored queries scored under seven fixed strategies. The retained typed-lexical strategy (S2) recovered 6/6 authored holdouts ($P = R = F_1 = 1.000$, realistic-tie $MRR = 0.944$, Brier = 0.131, Sem@3 = 1.000); the degree baseline recovered none. These are construction results: not OWL/SHACL conformance, semantic completeness, runtime retrieval, or evidence of any efficacy claim.

VI. PILOT RESULTS

A. Declared-Field Gate Conformance

The pilot loader admits a fixture set only when it isolates every implemented validator code exactly once alongside one

TABLE VI
FROZEN-PILOT CHECKS AND TERMINAL OUTCOMES (ROWS ARE NOT
COMPARABLE)

Check or terminal family	Exact outcome	Interpretation
Gate fixture agreement	13/13	authored-oracle agreement
Known-boundary acceptance	2/2	encoded limit, not a safety pass
Closed unknown-key rejection	1/1	proposal/replay parser parity
Detectable integrity-fault rejection	10/10	designated check operations
Known provenance-boundary acceptance	1/1	repair provenance unauthenticated
Adapter terminal classes	1 commit; 1 fallback; 5 failures / 7 assigned	complete classification of 7 assignments
Manifest-guard rejection	3/3	injected faults failed closed

Rows have different denominators and favorable directions; they must not be ranked as rates. Every value is an exact audit fact for designed fixtures.

valid control, so the 13/13 agreement and the observation of all 12 implemented error codes are *construction invariants of the harness*, not measured success rates. What the run adds is the part the loader does not enforce: each of the 12 isolated negative fixtures showed observed agreement with its authored expected code rather than reaching a different one, and every noncommit path preserved the complete canonical state. This is implementation conformance to an authored structured-field oracle over a single world state, not accuracy against independent semantic labels.

B. Repair-Arm Comparison on Designed Fixtures

The 12 initially invalid designed cases are partitioned into frozen repairability classes: 5 guided-repairable, 1 oracle-only, and 6 irreparable. Rejection-only and blind unchanged-candidate retry each committed 0/12. The counterexample-guided operator ρ committed 5/5 guided-repairable cases and, by design, 0/1 oracle-only and 0/6 irreparable cases; every ρ commit edited a mean of 1.0 candidate fields, and every failed arm execution left the canonical state unchanged. The state-reading reference callback committed 5/5 guided-repairable and 1/1 oracle-only cases, with 0/6 irreparable. These exact counts separate the three mechanisms on the designed fixtures: blind retry never converts a counterexample into a commit, ρ converts exactly the cases whose error payload carries sufficient information, and the oracle additionally converts the case requiring state knowledge. The 5 committed guided cases initially carried `POLICY_PRECONDITION_OMISSION+UNSATISFIED_PRECONDITION`, `POLICY_EFFECT_OMISSION`, `POLICY_EFFECT_VIOLATION`, `UNSATISFIED_PRECONDITION`, `POLICY_QUESTION_STAGE_EFFECT_VIOLATION`, respectively, and each became valid through the corresponding Table III edit $\delta(e)$ alone. This is operator conformance on frozen fixtures, not repair quality of any live model.

C. Integrity, Boundaries, and Assignment Accounting

All 10 faults prespecified as detectable were rejected by their designated check operations (10/10). Because this pi-

lot does not compare stable typed detector codes, it does not establish detector-layer attribution. Separately, the known provenance-boundary fixture substituted a different invalid precursor before the same recorded valid repair, recomputed the checksum, and passed replay as expected (1/1 undetected). Thus, the mechanism does not protect against a writer who can recompute unkeyed checksums, and replay does not authenticate the repair-generation operation. Two open boundary sentinels—unextracted narrative disclosure and simultaneous omission of a required object from candidate and policy—were intentionally accepted at the encoded layer (2/2), with 0 labelled as safety passes. The former unknown-top-level-field acceptance boundary was later reclassified as a closed negative regression: both proposal and replay parsers specifically rejected a complete 12-field candidate carrying one unknown key (1/1). This establishes parser rejection parity, not semantic safety. Among seven adapter/accounting assignments, outcomes were 1 commit, 1 symbolic fallback, and 5 adapter failures. Synthetic telemetry fields were populated and propagated for 2/7 assignments; these are schema-pinned constants verifying accounting-field propagation, not measurements. All 3/3 injected duplicate or missing-assignment guards failed closed. These are raw counts from synthetic offline fixtures.

D. Live Screening Pilot (Pilot-Only)

A separate screening pilot called the hosted proposer gpt-5.6-sol. It used 5 seed-indexed proposal calls per cell and repair budget $K = 1$; within each call, blind unchanged retry and ρ received the same initial candidate. The hosted revision was unpinned, token accounting was unavailable, and the seeds did not control the hosted sampler, so the calls are quasi-replicates rather than independent randomized samples. We report neither inferential statistics nor a model ranking.

In Table VII, a guided advantage appeared only in the policy-blind cell of the signal-repair state (Signal-v2), a state deliberately constructed to elicit guided-repairable errors. All 5/5 initial candidates were invalid; ρ committed 5/5, whereas blind retry committed 0/5. The other four current cells showed no guided advantage: three had already-valid initial candidates, and the frozen-base goal-directed cell produced irreparable quest-stage regressions. All 15/15 noncommit arm outcomes in the current cells preserved the prior-state hash. This supports mechanism transfer only in that error regime, not population efficacy, model promotion, or a general sample-efficiency claim. Accordingly, the two screening observations are pilot-only; the confirmatory transfer claim remains open. In the superseded v1 diagnostic, guided repair reduced up to 3 co-occurring errors to one irreparable knowledge error but still could not commit, showing that partial per-error repairs do not compose when any residual error is irreparable.

Tables V and VI report exact designed-fixture counts; neither supports a population or live-model estimate.

The reporting boundary is explicit: only rows admitted by the frozen fixture and its designated check operation enter the offline tables, whereas the live table arrives through its own hash-bound packet and never enters the offline denominator; Table IV lists what each lane leaves unmeasured.

TABLE VII
LIVE SCREENING PILOT, EXACT COUNTS (FIVE PROPOSALS PER CELL, $K=1$)

Base / condition	Initial valid	ρ vs blind commits	Δ	Initial errors
Frozen / visible	5/5	5/5 vs 5/5	+0	—
Frozen / blind	5/5	5/5 vs 5/5	+0	—
Frozen / goal-blind	0/5	0/5 vs 0/5	+0	QSR
Signal-v2 / visible	5/5	5/5 vs 5/5	+0	—
Signal-v2 / blind	0/5	5/5 vs 0/5	+5	PEO+PEV+PPO

QSR = quest-stage regression, PEO = policy-effect omission, PEV = policy-effect violation, PPO = policy-precondition omission. Δ is ρ commits minus blind commits for matched candidates. All counts are pilot-only, without inferential statistics; the superseded v1 diagnostic is excluded.

(a) Held proposal: the gate family, the unchanged state, the next valid entry, and the rule learned

```
[P] PROPOSAL ... Request disclosure of the sealed fact now
'Mira studies the dark tower and asks you to recover the signal lens first.'
[H] HELD - This request cannot be answered yet. The ledger defers it.
[V] GATE DISCLOSURE | state unchanged
[N] NEXT VALID ENTRY: Speak with Captain Mira at the end of the dock.
[V] RULE LEARNED | DISCLOSURE: Some facts stay sealed for good; the ledger never records them
```

(b) Committed proposal: entry, contribution clause, and unlocked affordance

```
[C] ENTRY #2 | COMMITTED - Signal lens installed. An authorized lead is now available.
[C] CONTRIBUTION #2 | Lead authorized; Harbor signal restored | STAGE 1>2 | CHAIN 2/3
[N] UNLOCKED | Ask Captain Mira for the authorized lead
```

(c) End-card receipt after the three commits

```
INVESTIGATOR'S CONTRIBUTION
#1 Signal lens secured | STAGE 0>1
#2 Lead authorized; Harbor signal restored | STAGE 1>2
#3 Tide-marks lead recorded | STAGE 2
RULES LEARNED 0 | none

ENTRIES 3 | HOLDS 0 | FINAL STAGE 2
VALIDATOR RECEIPT | STATE HASH 4b2310173dc05907...
HOLDS BY GATE | none
```

Fig. 3. The commit gate as the player reads it in the Godot slice: ledger crops from three scripted presentation stages of one authored fixture (engineering working captures, 1280×720; illustration only, not evidence). (a) A forbidden disclosure is *held*: the ledger names the gate family, states that the canonical state is unchanged, points to the next valid entry, and turns the hold into a rule the player can reuse. (b) A valid proposal is *committed*: the entry, a contribution clause derived only from the committed-snapshot delta (facts added, quest stage 1>2, case chain 2/3), and the affordance it unlocks. (c) The end card lists the three contribution clauses, the hold count, and the validator receipt whose terminal state hash (4b2310 . . .) is the one the replay reports. Every line is a presentation-only readout of committed snapshots (Fig. 1, lower strip); no oracle label, hidden fact, live model, or participant is involved, and the panels support no usability, immersion, or efficacy claim.

ENG1. A second implementation of the same gate, embedded in a Godot 4.7.1 slice, replayed the authored canonical trace: a held forbidden disclosure preserved the prior state hash and an authorized commit advanced it, matching the offline runner's terminal hash exactly. Fig. 3 illustrates what the player reads at those two outcomes and at the end of the episode: a hold names its gate family and the next valid entry, a commit appends a contribution clause and unlocks the next affordance, and the end card carries the hash-bearing receipt; the conformance evidence is the replayed trace and its deterministic evaluation, not these pixels. A case-file board, hint ladder, coach cards, and closing chain-assembly prompt read committed facts only; a companion dashboard shows per proposal the checks the engine ran, the one that held, the in-scope checks never reached, and the out-of-scope families, from a mirror that fails

closed on any disagreement with the engine’s codes. This is engine-local conformance of one scripted fixture (52/52 core checks plus a separately counted four-check extension). It carries no live model or Python transport, no participant input, and no narrative-quality result; the readout derives only from committed snapshots (Fig. 1, lower strip) and is not usability or efficacy evidence.

VII. DISCUSSION

A. Established Scope

The designed pilot establishes mechanism behavior only for its frozen cases: encoded checks isolate state, bounded repair and fallback follow their declared paths, named corruptions are rejected by their designated operations, and assigned rows remain classified. It does not attribute a rejection to a stable detector code.

The guided-versus-oracle boundary is the honest reading of the offline repair comparison: the state-reading callback converts the oracle-only case that ρ by construction cannot, while ρ matches the oracle on the guided-repairable class using only the validator’s counterexamples. The live extension separated ρ from blind retry only when the constructed state placed every proposal in the guided-repairable class; already-valid and guided-irreparable regimes were null, so regime dependence, the unpinned hosted revision, and quasi-replicated calls block any general sample-efficiency or model-capability claim. Feedback-consuming repair loops are preceded: Self-Refine feeds back language generated by the same untrusted model [45], Reflexion stores verbal self-reflections across trials [52], and Self-Debugging consumes execution results from a trusted full-program runtime [53]. ρ differs in channel and authority: its feedback is a typed error set computed by the deterministic validator before execution, and every failed live arm remains state-isolated. Its name follows counterexample-guided inductive synthesis, in which a verifier’s counterexample rather than the full specification state drives the next candidate [54]; here the verifier is the commit gate itself.

B. Relationship to Prior Work

TRACE-RPG complements rather than replaces constrained decoders [5], [6], [7], retrieval and memory components [8], [9], [10], and executable environments and benchmarks [1], [2], [22], [25]; process-linked traces instantiate AgentBoard’s measurement concern [23], not evidence of outperformance. The confirmatory studies—a preregistered multi-model comparison with an independent semantic oracle and a genuine blind-retry comparator, retrieval and memory ablations at matched encoded validity, and a blinded matched-validity human study with justified power and mixed-effects analysis—remain outside this paper; the screening pilot substitutes for none of them [38], [35], [43].

VIII. THREATS, ETHICS, AND ARTIFACT SCOPE

The evidence has nine explicit boundaries. **L1**: encoded predicates cannot detect semantics omitted from structured fields, and state consistency alone does not establish behavior

consistency [55]; narrative quality is an additional unmeasured construct. **L2**: one authored world and one constructed live state do not establish cross-world, cross-policy, or scale generalization. **L3**: the state-reading callback is an oracle; offline classes and regime-dependent screening do not establish general repair quality or sample efficiency. **L4**: the single hosted proposer has an unpinned revision, no token accounting, and uncontrolled seeds; no live Python–Godot round trip exists. **L5**: no participant or affect data exist; the game packet is engine-local conformance plus a measurement-ready protocol and supports no claim about comprehension, agency, enjoyment, immersion, or player benefit. **L6**: unkeyed hashes give integrity, not writer authentication or tamper resistance. **L7**: record consistency assumes one writer. **L8**: engine evidence is one Godot fixture with startup-dominated samples, not a portability or real-time result. **L9**: selected fixtures and screening cells support no population, safety-rate, or model-ranking inference. Internal validity also assumes independent policy expectations and no fixture leakage.

The pilot processes no participants or personal telemetry; future human and affect work requires ethics review, informed consent, minimization, retention rules, and opt-out. LLM judges, if used, remain auxiliary because automated and crowd judgements both require calibration and bias controls [38], [39], [40], [41].

IX. CONCLUSION

TRACE-RPG isolates candidate-event payloads from canonical mutation through an encoded symbolic commit gate and makes terminal outcomes auditable through linked records and state-semantic replay. The offline pilot establishes mechanism conformance on frozen fixtures; the live screening pilot separated ρ from blind retry (5/5 versus 0/5) only in the constructed repairable regime and remains pilot-only. Cross-model superiority, player-experience benefit, retrieval or memory effectiveness, affect efficacy, and commercial-engine performance remain confirmatory extensions. What already reaches the player is the gate itself: in the Godot slice a hold reads as a rule and a commit as a contribution, every line derived from committed snapshots.

DISCLOSURE OF AI USAGE

Large language models assisted prose, code and test drafting, and citation-identity checks; a hosted model produced the separately labelled live-screening candidates. Every reported count comes from deterministic runners or hash-bound receipts. The playable slice’s UI art (portrait, seals, panel textures, case-file board, evidence icons, guide vignette) and one prop mesh (the evidence desk) are curated AI-generated assets with per-file provenance; in Fig. 3 every state value, hash, and ledger line is engine-rendered from committed snapshots. The authors cross-checked claims against source artifacts and take full responsibility for the content.

DATA AND CODE AVAILABILITY

The repository contains the implementation, build sources, the frozen offline and live packets, the playable Godot slice

with its deterministic evaluation, and the machine-checked contribution, reference-topic, and experiment matrices from which Tables I and IV are generated. No reviewer archive or DOI deposit is claimed.

REFERENCES

- [1] M.-A. Côté, Á. Kádár, X. Yuan *et al.*, “TextWorld: A learning environment for text-based games,” in *Computer Games*, ser. Communications in Computer and Information Science. Springer International Publishing, 2019, vol. 1017, pp. 41–75.
- [2] R. Wang, P. Jansen, M.-A. Côté, and P. Ammanabrolu, “ScienceWorld: Is your agent smarter than a 5th grader?” in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2022, pp. 11 279–11 298.
- [3] N. Weir, R. Thomas, R. d’Amore, K. Hill, B. Van Durme, and H. Jhamtani, “Ontologically faithful generation of non-player character dialogues,” in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2024, pp. 9212–9242.
- [4] T. Ashby, B. K. Webb, G. Knapp, J. Searle, and N. Fulda, “Personalized quest and dialogue generation in role-playing games: A knowledge graph- and language model-based approach,” in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. ACM, 2023, pp. 1–20.
- [5] S. Geng, M. Josifoski, M. Peyrard, and R. West, “Grammar-constrained decoding for structured NLP tasks without finetuning,” in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2023, pp. 10 932–10 952.
- [6] L. Beurer-Kellner, M. Fischer, and M. Vechev, “Prompting is programming: A query language for large language models,” *Proceedings of the ACM on Programming Languages*, vol. 7, no. PLDI, pp. 1946–1969, 2023.
- [7] T. Scholak, N. Schucher, and D. Bahdanau, “PICARD: Parsing incrementally for constrained auto-regressive decoding from language models,” in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2021, pp. 9895–9901.
- [8] X. He, Y. Tian, Y. Sun *et al.*, “G-Retriever: Retrieval-augmented generation for textual graph understanding and question answering,” in *Advances in Neural Information Processing Systems*, vol. 37. Neural Information Processing Systems Foundation, 2024, pp. 132 876–132 907.
- [9] B. Jiménez Gutiérrez, Y. Shu, Y. Gu, M. Yasunaga, and Y. Su, “HippoRAG: Neurobiologically inspired long-term memory for large language models,” in *Advances in Neural Information Processing Systems*, vol. 37. Neural Information Processing Systems Foundation, 2024, pp. 59 532–59 569.
- [10] P. Chhikara, D. Khant, S. Aryan, T. Singh, and D. Yadav, “Mem0: Building production-ready AI agents with scalable long-term memory,” in *ECAI 2025*, ser. Frontiers in Artificial Intelligence and Applications, vol. 413. IOS Press, 2025, pp. 2993–3000.
- [11] J. S. Park, J. C. O’Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, “Generative agents: Interactive simulacra of human behavior,” in *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. ACM, 2023, pp. 1–22.
- [12] Y. Shao, L. Li, J. Dai, and X. Qiu, “Character-LLM: A trainable agent for role-playing,” in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2023, pp. 13 153–13 187.
- [13] N. Akoury, Q. Yang, and M. Iyyer, “A framework for exploring player perceptions of LLM-generated dialogue in commercial video games,” in *Findings of the Association for Computational Linguistics: EMNLP 2023*. Association for Computational Linguistics, 2023, pp. 2295–2311.
- [14] M. Hochreiter, S. Kriglstein, and G. Wallner, “Beyond pre-defined scripts: Player perceptions on generative non-player character dialogues,” in *Proceedings of the 31st International Conference on Intelligent User Interfaces*. ACM, 2026, pp. 2004–2018.
- [15] M. Yin, H. Zu, W. Gu *et al.*, “How contextualized generative AI shapes player experience in games,” *Entertainment Computing*, vol. 58, p. 101194, 2026, volume 58 (September 2026) assigned on the publisher record, verified 2026-09-03.
- [16] V. Figueiredo and D. Elumze, “Symbolically scaffolded play: Designing role-sensitive prompts for generative NPC dialogue,” 2025, preprint; no archival venue or publisher DOI verified as of 2026-09-03.
- [17] G. N. Yannakakis and J. Togelius, “Experience-driven procedural content generation,” *IEEE Transactions on Affective Computing*, vol. 2, no. 3, pp. 147–161, 2011.
- [18] D. Melhart, D. Gravina, and G. N. Yannakakis, “Moment-to-moment engagement prediction through the eyes of the observer: PUBG streaming on Twitch,” in *FDG ’20: Proceedings of the 15th International Conference on the Foundations of Digital Games*. ACM, 2020, pp. 1–10.
- [19] Z. Wang, Q. Guo, R. Singh, and X. Hu, “Do vision language models understand human engagement in games?” 2026, preprint; no archival venue or publisher DOI verified as of 2026-09-03.
- [20] L. Fan, G. Wang, Y. Jiang *et al.*, “MineDojo: Building open-ended embodied agents with internet-scale knowledge,” in *Advances in Neural Information Processing Systems*, vol. 35. Neural Information Processing Systems Foundation, 2022, pp. 18 343–18 362.
- [21] D. Pérez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, “General video game AI: A multitask framework for evaluating agents, games, and content generation algorithms,” *IEEE Transactions on Games*, vol. 11, no. 3, pp. 195–214, 2019.
- [22] X. Liu, H. Yu, H. Zhang *et al.*, “AgentBench: Evaluating LLMs as agents,” in *International Conference on Learning Representations*, 2024, pp. 52 989–53 046, peer-reviewed official proceedings record; no publisher DOI assigned.
- [23] C. Ma, J. Zhang, Z. Zhu *et al.*, “AgentBoard: An analytical evaluation board of multi-turn LLM agents,” in *Advances in Neural Information Processing Systems*, vol. 37. Neural Information Processing Systems Foundation, 2024, pp. 74 325–74 362.
- [24] X. Zhou, H. Zhu, L. Mathur *et al.*, “SOTOPIA: Interactive evaluation for social intelligence in language agents,” in *International Conference on Learning Representations*, 2024, pp. 40 975–41 019, peer-reviewed official proceedings record; no publisher DOI assigned.
- [25] D. Paglieri, B. Cupial, S. Coward *et al.*, “BALROG: Benchmarking agentic LLM and VLM reasoning on games,” in *International Conference on Learning Representations*, 2025, pp. 96 666–96 702, peer-reviewed official proceedings record; no publisher DOI assigned.
- [26] P. Le Pelletier de Woillemont, R. Labory, and V. Corruble, “Automated play-testing through RL based human-like play-styles generation,” *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 18, no. 1, pp. 146–154, 2022.
- [27] S. Kambhampati, K. Valmeekam, L. Guan *et al.*, “Position: LLMs can’t plan, but can help planning in LLM-Modulo frameworks,” in *Proceedings of the 41st International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 235. PMLR, 2024, pp. 22 895–22 907.
- [28] B. Ichter, A. Brohan, Y. Chebotar *et al.*, “Do as I can, not as I say: Grounding language in robotic affordances,” in *Proceedings of the 6th Conference on Robot Learning (CoRL 2022)*, ser. Proceedings of Machine Learning Research, vol. 205. PMLR, 2023, pp. 287–318.
- [29] H. Wang, C. M. Poskitt, and J. Sun, “AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents,” 2025, preprint (v3); the arXiv record states acceptance at ICSE 2026, but no proceedings DOI was resolvable as of 2026-09-03.
- [30] Z. Zeng, S. Li, J. Xi, A. Zhu, and P. Ammanabrolu, “Setting the DC: Tool-grounded D&D simulations to test LLM agents,” in *NeurIPS 2025 Workshop on Generative and Protective AI for Content Creation (GenProCC)*, 2025, non-archival workshop paper; OpenReview and its PDF list Zeng first and mark Zeng and Li as equal contributors, while NeurIPS and GenProCC venue lists retain a Li-first display; verified 2026-09-03.
- [31] E. Zhou, S. Basavatia, M. Siam, Z. Chen, and M. O. Riedl, “STORY2GAME: Generating (almost) everything in an interactive fiction game,” 2025, preprint; no archival venue or publisher DOI verified as of 2026-09-03.
- [32] S. Buongiorno, L. Klinkert, Z. Zhuang, T. Chawla, and C. Clark, “PANGeA: Procedural artificial narrative using generative AI for turn-based, role-playing video games,” *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 20, no. 1, pp. 156–166, 2024.
- [33] M. Vaucher, S. Silveira, S. Góngora, and L. Chiruzzo, “IVIE: A neuro-symbolic approach to incremental and validated generation of interactive fiction worlds,” in *Proceedings of the 17th International Conference on Computational Creativity (ICCC’26)*. Association for Computational Creativity, 2026, open-access archival proceedings published 2026-08-19 (ISBN only, no DOI), verified 2026-09-03; preprint arXiv:2606.13348.
- [34] S. Góngora, L. Chiruzzo, G. Méndez, and P. Gervás, “World-state transformations for neuro-symbolic interactive storytelling,” 2026, preprint; no archival venue or publisher DOI verified as of 2026-09-03.
- [35] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, and M. G. Bellemare, “Deep reinforcement learning at the edge of the statistical precipice,” in *Advances in Neural Information Processing Systems*, vol. 34. Curran Associates, Inc., 2021, pp. 29 304–29 320, peer-reviewed official proceedings record; no publisher DOI assigned.
- [36] J. Pineau, P. Vincent-Lamarre, K. Sinha *et al.*, “Improving reproducibility in machine learning research: A report from the NeurIPS 2019 reproducibility program,” *Journal of Machine Learning Research*, vol. 22, no. 164, pp. 1–20, 2021, peer-reviewed official journal article; no publisher DOI assigned.
- [37] P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, and J. Pineau, “Towards the systematic reporting of the energy and carbon footprints of machine learning,” *Journal of Machine Learning Research*, vol. 21, no. 248, pp. 1–43, 2020, peer-reviewed official journal article; no publisher DOI assigned.
- [38] C. van der Lee, A. Gatt, E. van Miltenburg, S. Wubben, and E. Krahmer, “Best practices for the human evaluation of automatically generated text,” in *Proceedings of the 12th International Conference on Natural Language Generation*. Association for Computational Linguistics, 2019, pp. 355–368.
- [39] M. Karpinska, N. Akoury, and M. Iyyer, “The perils of using Mechanical Turk to evaluate open-ended text generation,” in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2021, pp. 1265–1285.
- [40] L. Zheng, W.-L. Chiang, Y. Sheng *et al.*, “Judging LLM-as-a-judge with MT-Bench and Chatbot Arena,” in *Advances in Neural Information Processing Systems*, vol. 36. Neural Information Processing Systems Foundation, 2023, pp. 46 595–46 623.
- [41] G. H. Chen, S. Chen, Z. Liu, F. Jiang, and B. Wang, “Humans or LLMs as the judge? a study on judgement bias,” in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2024, pp. 8301–8327.
- [42] D. Lakens, A. M. Scheel, and P. M. Isager, “Equivalence testing for psychological research: A tutorial,” *Advances in Methods and Practices in Psychological Science*, vol. 1, no. 2, pp. 259–269, 2018.
- [43] D. J. Barr, R. Levy, C. Scheepers, and H. J. Tily, “Random effects structure for confirmatory hypothesis testing: Keep it maximal,” *Journal of Memory and Language*, vol. 68, no. 3, pp. 255–278, 2013.
- [44] E. Feng, W. Zhou, Z. Liu *et al.*, “Get experience from practice: LLM agents with record & replay,” 2025, preprint; no archival venue or publisher DOI verified as of 2026-09-03.
- [45] A. Madaan, N. Tandon, P. Gupta *et al.*, “Self-Refine: Iterative refinement with self-feedback,” in *Advances in Neural Information Processing Systems 36 (NeurIPS)*, 2023, pp. 46 534–46 594.
- [46] M. O. Riedl, C. J. Saretto, and R. M. Young, “Managing interaction between users and agents in a multi-agent storytelling environment,” in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2003, pp. 741–748.
- [47] R. E. Fikes and N. J. Nilsson, “STRIPS: A new approach to the application of theorem proving to problem solving,” *Artificial Intelligence*, vol. 2, no. 3–4, pp. 189–208, 1971.
- [48] F. B. Schneider, “Enforceable security policies,” *ACM Transactions on Information and System Security*, vol. 3, no. 1, pp. 30–50, 2000.
- [49] T. Haerder and A. Reuter, “Principles of transaction-oriented database recovery,” *ACM Computing Surveys*, vol. 15, no. 4, pp. 287–317, 1983.
- [50] R. Evans and E. Short, “Versu—a simulationist storytelling system,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 2, pp. 113–130, 2014, predecessor title of IEEE Transactions on Games.
- [51] S. G. Ware and C. Siler, “Sabre: A narrative planner supporting intention and deep theory of mind,” in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, vol. 17, no. 1, 2021, pp. 99–106.
- [52] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, “Reflexion: Language agents with verbal reinforcement learning,” in *Advances in Neural Information Processing Systems 36 (NeurIPS)*, 2023, pp. 8634–8652.
- [53] X. Chen, M. Lin, N. Schärli, and D. Zhou, “Teaching large language models to self-debug,” in *The Twelfth International Conference on Learning Representations (ICLR)*, 2024, accepted poster; the OpenReview forum is the archival record (no publisher DOI), verified 2026-09-03; preprint arXiv:2304.05128.
- [54] A. Solar-Lezama, L. Tancu, R. Bodik, S. Seshia, and V. Saraswat, “Combinatorial sketching for finite programs,” in *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2006, pp. 404–415.
- [55] Y. Huang, G. Chen, J. Yao *et al.*, “Beyond state consistency: Behavior consistency in text-based world models,” 2026, preprint; no archival venue or publisher DOI verified as of 2026-09-03.