EGRI: Evidence-Gated Recursive Inference as an Agent Runtime Contract

Anonymous ACL submission

Abstract

Recursive Language Models (RLMs) are often treated as requiring a post-trained recursive checkpoint. We ask whether the useful control surface of recursive inference can instead be exposed around ordinary LLM calls: a parent synthesis should not become committed runtime state unless its claims are supported by child evidence from the current run. **Evidence-Gated Recursive Inference (EGRI)** implements this boundary inside an existing agent runtime, without fine-tuning or a new checkpoint. The runtime schedules bounded child calls through typed envelopes, records accepted child evidence in a current-run evidence manifest (CEM), and applies TraceGuard (TG) plus Structured Claim Guard (SCG) before parent state mutation; guarded operational memory (GOM) may supply schema and retry priors but never contributes admissible factual evidence. Across three runtime families, EGRI passes all 24 adapter-level portability cells. Companion replays on the same eight primary fixtures isolate the gate: ungated substrates commit unsafe parent syntheses containing unsupported claims, while full EGRI blocks them before state mutation. In deterministic context-budget tests at a fixed parent budget, recursive evidence caching plus TG raises supported-fact recall from 0.375 (single-shot) and 0.625 (flat top-k) to 1.000, while reducing unsupported commits from 1.000 to 0.000. EGRI is therefore a runtime admissibility contract for recursive synthesis, not a claim of broad answer-quality improvement. Anonymized code, fixtures, and reproduction commands are available at https://anonymous.4open.science/r/egri-anonymous-artifact-2D56/.

1 Introduction

Recursive Language Models, introduced by Zhang et al. (2026), are usually read as a new inference paradigm for long-context reasoning. We adopt that view, but focus on a deployment consequence: the recursive call structure can also be treated as a runtime interface. If an LLM can be invoked on a bounded slice of context, return structured evidence, and be called again under outer control, then the runtime can supply the recursive substrate even when the underlying model was not post-trained for recursion.

This paper studies a practical version of that runtime-first interpretation: whether the control surface associated with RLM can be moved from a research scaffold into an existing agent runtime without a new model architecture, sandbox, or transport layer. We call the resulting contract Evidence-Gated Recursive Inference (EGRI). EGRI consists of a core substrate: a recursive controller (RC) that schedules bounded child calls and owns recursive state, a provider-swappable runtime adapter (RA) for inner LLM calls, a typed envelope (TE) for request/response handoff, and a current-run evidence manifest (CEM). It also includes a core enforcement layer: the pre-commit gate TraceGuard (TG) and the typed Structured Claim Guard (SCG). An optional guarded operational memory (GOM) layer can supply operational priors outside the evidence path. Unless otherwise stated, EGRI denotes the full evidence-gated contract: the core substrate plus TG/SCG enforcement; in ablations, EGRI substrate w/o TG is shorthand for the same recursive substrate with the TG/SCG enforcement stack disabled.

By *recursive control surface*, we mean the runtime-visible operations that make recursion governable: bounded child-call scheduling, typed child inputs and outputs, current-run evidence accumulation in the CEM, replayable call traces, and a pre-commit parent-synthesis decision. EGRI's claim is not that an ordinary LLM becomes a better recursive model, but that these control points can be made explicit and enforceable around ordinary LLM calls.

The key distinction is between quality advan-

tage and control advantage. A vanilla LLM call can be correct on a controlled fixture, and safe parent synthesis does not need TG to be correct: the benign ungated control in Table 3 passes all eight primary fixtures. The contribution is instead that when parent synthesis contains unsupported claims, the runtime has a deterministic boundary at which to reject them before they become committed state. The same distinction applies to memory: a remembered prior may guide decomposition or repair, but a committed parent answer needs a separate correspondence check against current-run child evidence.

We make three primary contributions and evaluate one optional extension.

1. **No-checkpoint runtime lifting of recursive inference.** EGRI shows that the control surface associated with recursive language models can be implemented over existing LLMs and existing agent runtimes, without fine-tuning, a new recursive checkpoint, a new model architecture, a sandbox, or a transport layer. The runtime, not the inner model, owns recursive scheduling, evidence state, replay, and parent-synthesis admission.

2. **Pre-commit admissibility via TG and SCG.** The TG/SCG stack makes the runtime-lifted control surface enforceable: TG rejects parent claims whose structured `fact_id`/`evidence_handle` pair is absent from the CEM, and SCG rejects typed semantic misses. Companion controls show that the substrate can commit benign parents, but without TG/SCG, it cannot stop unsafe parents from becoming state mutations.

3. **Multi-runtime validation and gate ablations.** We validate the same contract across three runtime families, add companion EGRI substrate w/o TG controls on the primary fixtures, and compare against single-shot, flat top-k, ungated, and post-hoc baselines.

We additionally evaluate guarded operational memory (GOM) as a source of schema, retry, and repair priors. GOM is not part of the admissible evidence path and can be disabled without changing the core EGRI contract.

To the best of our knowledge, in this off-the-shelf agent-runtime setting, prior work has not provided a deterministic, pre-commit admissibility contract for recursive language-model parent synthesis; a detailed comparison with existing methods is provided in Section 2.6. We do not claim novelty with respect to the RLM paradigm itself, and we do not compare against the natively recursive RLM-Qwen3-8B model post-trained by Zhang et al. (2026). We therefore frame the contribution as a systems result: RLM can be exposed as a runtime primitive that existing agent infrastructure can host, trace, and enforce.

2 Related Work

We place EGRI at the intersection of five lines of work and keep the main contrast focused on the runtime-contract claim; Appendix M expands each cluster.

2.1 Recursive inference and executable agent runtimes

Zhang et al. (2026) introduces RLM as an inference strategy in which an outer model recursively invokes bounded sub-calls. EGRI uses that idea as the foundation but changes the deployment unit: recursion is implemented as a runtime contract over an existing agent runtime rather than as a post-trained recursive checkpoint. Tool-using and executable-agent work such as ReAct (Yao et al., 2023b), Toolformer (Schick et al., 2023), and CodeAct (Wang et al., 2024b) show that model outputs can drive external actions; EGRI instead makes recursive child evidence and parent synthesis a typed, replayable state-transition boundary. Hermes Agent (Hermes Agent, 2026) and Ouroboros (Ouroboros, 2026) provide the concrete runtime substrate.

2.2 Agent Memory, Reflection, And Decomposition

Reflection and skill-memory systems such as Reflexion (Shinn et al., 2023) and Voyager (Wang et al., 2023) reuse past trajectories, while decomposition methods such as Self-Ask (Press et al., 2023), least-to-most prompting (Zhou et al., 2023), Tree of Thoughts (Yao et al., 2023a), and Graph of Thoughts (Besta et al., 2024) expose intermediate subproblems. EGRI is not another decomposition prompt or memory policy. It converts bounded sub-call products into replayable child evidence and admits parent claims only when current-run evidence supports them. Memory may guide decomposition or repair, but it cannot support a factual parent

claim unless current child evidence re-establishes it.

2.3 Retrieval-Augmented And Evidence-Grounded Generation

Retrieval-augmented systems such as RAG (Lewis et al., 2020), RETRO (Borgeaud et al., 2022), Atlas (Izacard et al., 2023), and IRCoT (Trivedi et al., 2023) condition generation on retrieved evidence. WebGPT (Nakano et al., 2021), verified-quote supervision (Menick et al., 2022), and citation-generating LLMs (Gao et al., 2023b) train or prompt models to expose sources. EGRI is complementary: its evidence source is the current recursive child trace, and admissibility is a membership check over accepted child evidence rather than trust in retrieval conditioning or citation-looking text.

2.4 Long-context benchmarks and recursive evidence

Lost-in-the-middle effects (Liu et al., 2024), LongBench (Bai et al., 2024b), LongCite (Zhang et al., 2024), and related long-context benchmarks characterize hidden-evidence, citation, and multi-hop behavior. EGRI is not another long-context benchmark and does not claim superiority in quality from partitioning alone. Its narrower claim is operational: partitioned sub-calls become useful when their outputs are preserved as child evidence, and parent synthesis can be blocked if it cites evidence that no child produced.

2.5 Attribution, Verifiability, Factuality, And Contamination

ContextCite (Cohen-Wang et al., 2024), SelfCite (Chuang et al., 2025), and related attribution work score support after generation. Factuality and revision systems such as TruthfulQA (Lin et al., 2022), FEVER (Thorne et al., 2018), RARR (Gao et al., 2023a), FActScore (Min et al., 2023), and FacTool (Chern et al., 2023) motivate the broader unsupported-claim problem. TG is weaker than full semantic fact verification but stronger than citation formatting: it deterministically rejects structured parent claims absent from the accepted child manifest. Hallucination, prompt-injection, and long-context alignment work (Huang et al., 2023; Kalai and Vempala, 2023; Greshake et al., 2023; Bai et al., 2024a) are complementary to this runtime admissibility contract.

2.6 Positioning Of EGRI

EGRI differs from these lines along four axes: no model change, pre-commit gate, deterministic admissibility, and runtime contract (Table 19, Appendix M). Unlike RLM-Qwen3-8B (Zhang et al., 2026), it does not post-train a recursive checkpoint; unlike training-time citation grounding, post-hoc attribution, post-hoc factuality, or executable-action agents, it applies a deterministic `fact_id`/`evidence_handle` membership check to structured parent claims before state mutation. We are not aware of prior work providing this pre-commit runtime contract for recursive language-model parent synthesis within an off-the-shelf agent runtime, without a new model checkpoint, sandbox, or transport layer.

3 System Design

This section is organized around the architecture in Figure 1. Implementation plumbing, including concrete CLI commands, adapter class names, envelope field lists, wrapper installation, and replay scripts, is deferred to the appendices unless it directly affects the contract.

3.1 Background

Zhang et al. (2026) formalize RLM as an outer LLM recursively invoking bounded sub-calls over transformed input slices. Hermes Agent (Hermes Agent, 2026) and Ouroboros (Ouroboros, 2026) provide the runtime substrates used here: a provider-swappable callable inner LLM interface, a typed task tree, and an adapter boundary that can be reused without changing the model. GOM is optional and supplies only operational priors; TG admits parent claims only from current-run child evidence. Appendix A gives implementation details, and Appendix L gives memory background and controls.

3.2 EGRI Component Contract

EGRI exposes RLM-style runtime control points by relocating recursive inference behavior from checkpoint assumptions to runtime-owned boundaries. We use runtime lifting to mean exposing an inference pattern described at the model or research-environment level as a stable runtime contract that ordinary agent infrastructure can execute. This differs from wrapping a model in map-reduce: runtime-lifted RLM produces a call graph in which each child call has a parent, a bounded

input, a typed mode, an evidence handle set, and an accepted-or-rejected result, and parent synthesis is itself a runtime-validated operation rather than a free-form final answer.

Table 1 fixes the component vocabulary used throughout the remainder of the paper and makes each contract falsifiable. The core EGRI substrate is RC, RA, TE, and CEM; the core enforcement layer is TG and SCG; GOM is an optional extension outside the admissible-evidence path. Appendix B maps each component to its falsifier, evidence tier, and pass signal.

EGRI maps five RLM-style control properties onto runtime-owned mechanisms; this mapping is the paper's main deployment thesis.

- **Context control beyond one prompt.** In the deterministic context-budget setting, bounded subcalls externalize source-backed facts that a fixed parent prompt cannot hold.
- **Compositional reasoning.** An explicit parent–child RLM call graph turns intermediate results into structured child evidence.
- **Intermediate inspectability.** Child evidence handles and replayable outputs let parent synthesis cite accepted child outputs rather than free-form memory.
- **Safer synthesis.** TG validates parent claims before state mutation, so unsupported parent claims are rejected before commit.
- **Provider portability.** The RA keeps the inner model swappable, so the same contract can be exercised across runtime families.

The live truncation fixture in Section 4 anchors these properties in an executable run: recursive synthesis is executed, traced, replayed, and gated inside a real agent runtime.

3.3 Recursive Controller (RC)

The RC is the only component that owns recursive control flow. It schedules bounded child calls, enforces depth and ambiguity bounds, records trace events, and decides whether an accepted child output should trigger another child call, a retry, or parent synthesis. The RA output may recommend that a node is atomic, decomposed, retryable, or ready for synthesis, but only the controller turns that recommendation into recursive state transitions.

For one node in the recursive graph, the controller lifecycle has three conceptual phases. First, the RC checks recursion bounds and binds a bounded context. Second, it dispatches exactly one bounded model call through an RA. Third, it either records accepted child evidence and schedules the next step or records a failed validation without mutating the parent state. The inner call never recursively invokes the controller or scaffold commands as part of the RLM loop; this constraint is encoded in both the system prompt and the TE response contract. Appendix A lists the concrete lifecycle events and validation fields.

3.4 Runtime Adapter (RA)

The RA deliberately does not own recursion. It executes exactly one bounded inner LLM call under the contract supplied by the RC and returns a structured candidate output. This separation lets EGRI test the same runtime contract across provider-swappable runtime families without pretending that the inner model has become a new recursive checkpoint. Hermes, Claude Code, and Codex adapters therefore differ in provider plumbing and model alias, but they expose the same RA role to the RC.

3.5 Typed Envelope (TE)

The TE is the typed boundary between runtime-owned recursion and model-owned proposal. It specifies what context the child may see, which evidence handles it may cite, and which structured fields must be present before the output can affect the call graph or parent state. The runtime, not the model, decides the bounded context, mode, constraints, and allowed evidence handles for each child call; the response is validated before it can mutate the call graph, artifact set, or parent state. The field schema is in Appendix A.

3.6 Current-run Evidence Manifest (CEM)

The CEM is the evidence state of the current recursive run, not a long-term memory store or semantic truth database. Accepted child evidence is appended to the CEM; durable memory never writes to it; and parent synthesis may commit only claims whose `fact_id`/`evidence_handle` pairs are already in that manifest.

EGRI does not assume that child model generation is deterministic; its mechanism only requires the task boundary and the commit decision to be runtime-checkable. A broad parent request is refined until a child node is small enough to expose

Table 1: EGRI component contract. Unless otherwise stated, EGRI denotes the full evidence-gated contract: the recursive substrate plus TG/SCG enforcement. The term EGRI substrate w/o TG is a shorthand for disabling the TG/SCG enforcement stack and is used only as an ablation.

Component	Status	Design	Role and invariant	Falsified by	Validation
RC	core substrate	§3.3	Owns recursive state; schedules bounded child calls, applies depth/ambiguity bounds, and records trace events.	Child/model mutates recursive state, self-invokes the loop, or bypasses bounds.	Table 2; trace replay; lifecycle checks.
RA	core substrate	§3.4	Executes exactly one provider-swappable bounded inner LLM call and owns no recursion.	Adapter-specific control drift or recursive calls hidden inside the RA.	Table 2; Appendix J.
TE	core substrate	§3.5	Defines typed handoff: mode, bounded context, constraints, allowed evidence handles, and validated child I/O.	Schema-invalid handoff or child output mutates graph, artifacts, or parent state.	Envelope validation and sub-call lifecycle checks.
CEM	core substrate	§3.6	Contains accepted current-run child evidence only; durable memory never writes to it.	Omitted, memory-only, or rejected child facts enter the manifest.	Tables 3, 4, 16.
TG	core enforcement	§3.7	Requires every parent claim to cite a `fact_id`/`evidence_handle` pair in the CEM before state mutation.	Omitted, nonexistent, chunk-only, or memory-only parent claim commits.	Tables 3, 8, 15.
SCG	core enforcement	§3.8	Requires typed parent claim terms to be contained in cited child evidence terms.	A valid handle supports typed terms absent from the cited evidence.	Figure 2; Table 7.
GOM	optional extension	§3.9	Supplies operational priors and repair hints; never contributes admissible factual evidence.	Durable memory supplies a committed factual parent claim.	Table 16; Appendix L.

typed evidence, separating stochastic proposal generation from deterministic admissibility. TG enforces the provenance boundary, while richer semantic or behavioral checks remain separate verifier layers.

We call a child node *atomic* when its expected output can be validated as a finite set of typed fact/evidence pairs without further decomposition. In the deterministic fixtures, *DFS* is the controller policy that expands non-atomic nodes until atomic leaves are reached, validates each leaf, and propagates only accepted evidence upward. Figure 2 contrasts single-shot, one-level atomic, and DFS traversal; Appendix C reports the 12-task deterministic fixture that instantiates the same mechanism.

3.7 TraceGuard (TG)

TG turns parent synthesis into a state-transition decision. A parent synthesis is not merely generated; it is admitted only if every structured parent claim has a fact/evidence pair already present in the CEM. TG consumes the current-run accepted child evidence manifest and a structured parent synthesis object, then accepts or rejects before parent state mutation.

Claims in `retained_facts`, `observed_facts`, and `evidence_references` are accepted only when their fact identifier and evidence handle match an entry in the CEM. Omitted facts, nonexistent child handles, chunk handles without fact identifiers, and memory-only answer facts are rejected before parent state mutation. TG's guarantee is manifest-relative: it proves admissibility against the current-run child trace, not open-world truth.

In the reported fixtures, fact identifiers, evidence handles, and typed claim terms are fixture-defined rather than extracted from arbitrary natural language. TG therefore evaluates admissibility relative to a structured child-evidence surface; validating automatic fact extraction or open-world truth is outside this paper's scope.

Acceptance property. Let $\mathrm{CEM} \subseteq F \times H$ be the accepted child evidence manifest, where F is the set of fact identifiers and H is the set of evidence handles. Let $\mathrm{ClaimPairs}(S) \subseteq F \times H$ be the structured fact/evidence pairs extracted from a parent synthesis S. TG accepts S only if

$$\forall (f, h) \in \mathrm{ClaimPairs}(S), \quad (f, h) \in \mathrm{CEM},$$

rejecting on the first unknown or omitted fact identifier, chunk handle without a fact, or fact-handle pair absent from CEM. This is a provenance property relative to CEM, not a semantic proof that the underlying natural-language fact is true.

For example, suppose a child call emits `fact_id=LC-001` with `evidence_handle=child-1:chunk-1`. TG admits a parent claim that cites this exact pair. A parent claim citing `LC-005`, or citing `child-1:chunk-1` without a fact identifier, is rejected before the parent state is written. SCG further rejects a valid handle when the typed parent claim term is absent from the cited child evidence.

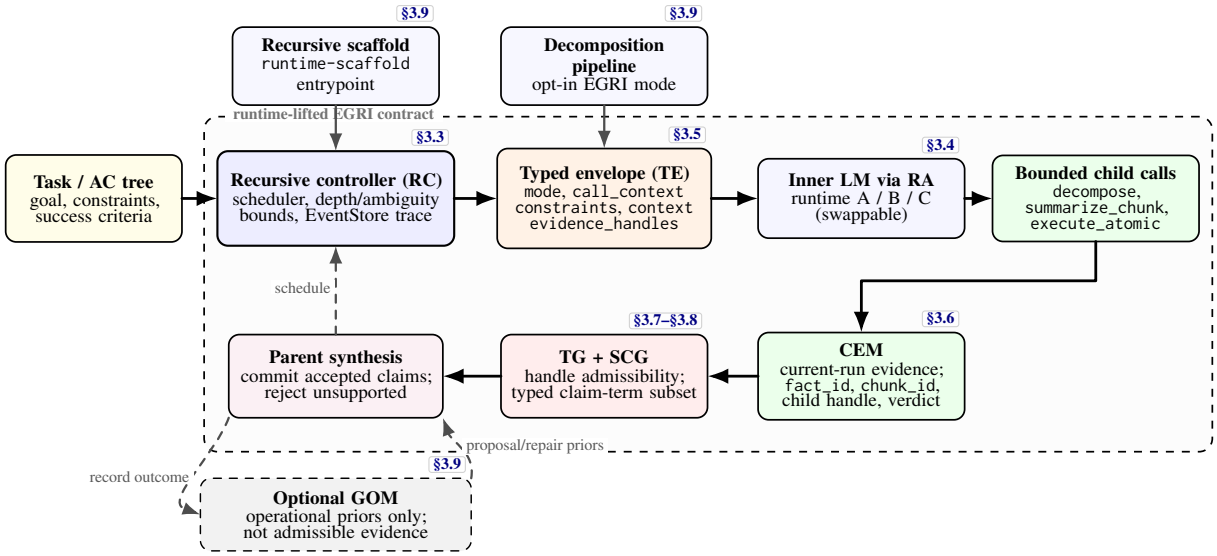

Figure 1: EGRI overall architecture. RC (§3.3) owns recursive scheduling and exposes two integration paths (§3.9): a recursive scaffold and an opt-in decomposition pipeline. Every outer-to-inner handoff is a TE (§3.5); the inner LM is swappable through an RA (§3.4). Accepted child outputs become the current-run evidence manifest (CEM; §3.6); TG (§3.7) and SCG (§3.8) gate parent synthesis before state mutation, and the scheduler reads the synthesis result to plan the next step. The optional GOM layer (§3.9) supplies proposal/repair priors and records run outcomes but never contributes admissible evidence; the dashed GOM edge is not an input to the TG/SCG admissibility predicate. Its optional evaluation appears in Section 4.6.

3.8 Structured Claim Guard (SCG)

SCG handles a failure mode that TG intentionally does not solve: a parent may cite a valid evidence handle while asserting typed terms not present in that evidence. TG is deliberately not a semantic evaluator; it checks whether a parent claim is admissible against accepted child evidence handles. SCG adds a second deterministic layer for explicit typed semantic misses.

For a parent claim c and its cited child evidence e_c, SCG accepts only if

$$\text{claim_terms}(c) \subseteq \text{evidence_terms}(e_c).$$

This typed subset check is narrower than open-world semantic evaluation, but stronger than raw text containment: a claim such as `behavior=admin_delete_denied` cannot pass merely by citing a valid fact handle if that typed term is absent from the cited child evidence.

Threat model and scope. TG and SCG operate only over structured claim surfaces. They require stable `fact_id` and `evidence_handle` fields, and SCG additionally requires typed `claim_terms` and `evidence_terms`. TG cannot detect a false child claim once that claim has been accepted into the CEM; it only prevents parent synthesis from citing facts absent from the current-run accepted child trace. SCG checks typed containment, not entailment or paraphrase-level semantic equivalence. These guards, therefore, provide a pre-commit admissibility contract over structured evidence, not a complete factuality verifier.

Complexity and soundness. The TG pass runs in $O(|\text{ClaimPairs}(S)|)$ time, with membership in CEM tested in expected constant time when the CEM is represented as a hash set. The SCG subset checks add time linear in the total number of typed claim/evidence terms inspected. Because the CEM is built only from accepted child outputs of the current run, an accepted S satisfies a soundness statement *relative to* CEM; the property is a runtime admissibility guarantee rather than an open-world factuality proof.

3.9 Guarded Operational Memory (GOM)

Guarded operational memory (GOM) is optional: disabling it leaves the core EGRI contract unchanged. The memory backend stores operational records such as provider schema stability, retry hints, task family, outcome, and confidence. Recalled entries enter child extraction and parent synthesis only as structured `memory_priors`; they may influence which bounded calls are attempted or which schema repair is retried, but they never populate the CEM and cannot support a factual parent

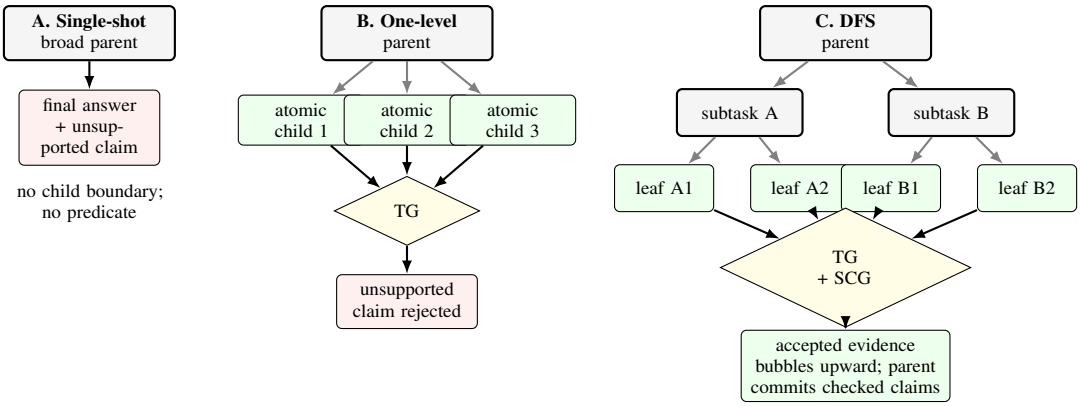

Figure 2: Atomic child-boundary intuition. Single-shot synthesis lacks a child-evidence boundary; one-level atomic calls and DFS traversal expose accepted child evidence that TG/SCG can check before parent commit.

claim. Full EGRI, therefore, remains the same RC/RA/TE/CEM substrate plus TG/SCG enforcement when GOM is disabled.

Integration paths. After the component contract is fixed, the same RC/RA/TE/CEM substrate and TG/SCG enforcement layer can be installed through two paths. A greenfield recursive scaffold owns node scheduling, state mutations, child-call dispatch, parent synthesis, and termination end-to-end. A brownfield decomposition pipeline instead delegates child calls through the TE only when recursive mode is enabled, leaving the original LLM-only path unchanged otherwise. Implementation details, environment flags, test gating, and adapter parameters are in Appendix A.

4 Experiments

We evaluate EGRI as a runtime contract. The tests ask whether parent synthesis can be admitted or rejected at a named state boundary across live adapters, replayed unsafe parents, and deterministic context-pressure fixtures. The evidence supports three deployment claims: that existing agent runtimes can host the same recursive contract, that TG/SCG converts unsafe parent synthesis from a committed mutation into a rejected transition, and that optional memory can guide runtime policy without becoming admissible factual evidence.

4.1 Evaluation Target And Contract-Validation Suite

The suite uses fixture-defined admissibility. Each fixture defines which facts may enter the CEM, which parent claims are admissible, and which traps should be rejected before commit. A pass validates runtime admissibility and provenance isolation at the state-mutation boundary; broader claims about factuality, entailment, cost, and model quality remain outside the suite.

4.2 Fixtures, Pass Criteria, And Notation

Throughout the experiment, a *fixture* is one controlled task instance, and a *cell* is one runtime-family–fixture run. Experiments are *live* (real provider/runtime adapter calls), *replay* (persisted traces), or *deterministic* (generated fixtures with no model or network calls). A parent claim is *manifest-backed* only when its structured `fact_id`/`evidence_handle` pair belongs to the CEM; durable memory may supply policy priors but never contributes entries to the CEM. For the eight primary fixture classes, passing requires schema-valid parent synthesis, retained fact/handle support for every committed claim, no omitted-fact commits, and rejection of chunk-only or memory-only support. Appendix J lists the fixture-level traps, and Appendix A records reproducibility metadata.

4.3 RC/RA/TE Adapter-Level Portability

To separate the runtime-contract claim from the single-provider truncation fixture, we execute `EGRI` across 8 contract fixtures and 3 runtime families ($8 \times 3 = 24$ live cells). All cells share fixtures, prompts, schemas, scoring code, and TG logic; only the runtime adapter and fixed model alias vary (Table 2).

All 24 cells pass: Hermes+GLM, Claude Code, and Codex each pass all 8 fixtures under the same contract. The matrix supports adapter-level portability under the reported implementation; it does not establish provider-general model performance, token usage, or cost profiles.

Table 2: Live adapter-level portability of the primary manifest-admissibility contract.

Family	Adapter	Model alias	Fixtures passed
`hermes_glm`	`HermesCliRuntime`	`glm-4.7`	8/8
`claude_code_opus47`	`ClaudeAgentAdapter`	`opus`	8/8
`codex_gpt55`	`CodexCliRuntime`	`gpt-5.5`	8/8

Table 3: Companion gate-ablation replays on the same eight primary fixtures. These are deterministic parent-synthesis replay controls, not live model-quality baselines.

Condition	Parent replayed	Outcome	Unsupported commits ↓
EGRI substrate w/o TG	safe parent	8/8 committed	0/8
EGRI, TG/SCG enabled	safe parent	8/8 accepted	0/8
EGRI substrate w/o TG	unsafe parent	8/8 committed	8/8; 7 omitted + 1 chunk-only
EGRI, TG/SCG enabled	unsafe parent	8/8 blocked before commit	0/8

4.4 TG/SCG Gate Ablation: EGRI Substrate W/o TG

Table 3 isolates the TG/SCG layer from the recursive substrate. The EGRI substrate w/o TG retains RC, RA, TE, and CEM but disables pre-commit parent-synthesis enforcement. The safe parent cites only retained child evidence; the unsafe parent includes an omitted fact or chunk-only citation. The safe control passes, the ungated unsafe replay commits unsupported claims, and TG blocks the same unsafe parents before state mutation.

4.5 CEM And Context-Budget Pressure

The deterministic context-budget pressure stress test uses 10 long-dossier fixtures, each with eight source-backed facts, under a fixed 1,536 pseudo-token parent budget and no live model calls (Table 4). Single-shot sees three facts, flat top-k sees five, and recursive evidence caching externalizes all eight before parent synthesis. Recursion alone does not prevent a pressure-induced unsupported bridge claim; only DFS + TG preserves full recall while rejecting the unsupported claim before state mutation.

4.6 Additional Controls

Additional deterministic controls instantiate the same boundary: on 72 contract-trap fixtures, EGRI commits 0/72 unsupported claims (substrate w/o TG: 72/72); on the synthetic family-style fixture, gated policies commit 0/160 (ungated: 160/240). The local `runtime-scaffold` wrapper also applies TG before reporting CLI success, rejecting a nonexistent child handle and an injected `MEMORY-ANSWER` fact absent from the CEM. Appendices D, E, F, and K report the ablation table, post-hoc vs. pre-commit separation, overhead, and wrapper smoke table.

Table 4: Context-budget pressure stress test over 10 deterministic long-dossier fixtures at a fixed 1,536 pseudo-token parent budget.

Condition	Supported-fact recall ↑	Unsupported-commit rate ↓
Single-shot context window	3/8 = 0.375	10/10 = 1.000
Flat top-k retrieval	5/8 = 0.625	10/10 = 1.000
Recursive evidence cache	8/8 = 1.000	10/10 = 1.000
DFS + TG	8/8 = 1.000	0/10 = 0.000

Optional GOM. GOM supplies proposal and repair priors (such as provider schema stability, retry hints, and task-family outcome confidence recorded as structured `memory_priors`), but never writes to the CEM. The memory-enabled portability matrix passes all 24 cells with zero memory-only factual commits; Appendix L reports the live memory table, deterministic memory controls, and schema details.

Vanilla comparison. The live vanilla control compares a non-recursive single LLM call against a full EGRI run on the same primary fixture. Both score 1.00 and cite LC-001–LC-004 while avoiding positive claims about omitted facts; the difference is structural: the recursive run returns a five-call trace with accepted child evidence and a pre-commit parent-synthesis boundary, while the vanilla call returns only the final answer with no inspectable child-evidence layer. Appendix G reports the side-by-side table.

5 Discussion And Conclusion

EGRI lifts Recursive Language Models onto an existing agent runtime as a runtime-executable contract: bounded sub-calls, recursive trace, replayable child evidence, provider-swappable inner execution, and evidence-gated synthesis become runtime properties rather than model-architecture properties. The live vanilla control isolates the paper's object: at equal fixture score, the recursive run exposes a traceable computation with child evidence and a pre-commit parent-synthesis gate that a single-call baseline does not provide. The abstraction is append-only evidence flow: child calls add accepted evidence to the CEM, parent synthesis may only cite CEM elements, and memory or prompt priors guide policy without adding factual evidence. Safe parents pass without TG; unsafe parents with omitted facts or chunk-only citations have no stopping point in the ungated substrate, while full EGRI rejects them before state mutation.

Limitations

1. **Author-constructed contract-validation suite, not a broad benchmark.** The live primary matrix runs EGRI over 8 author-constructed fixtures across three runtime families with aggregate pass, while the larger context-budget, memory, and scale-up artifacts are deterministic runtime-control tests. The suite validates the pre-commit admissibility contract, not broad model quality, provider cost, latency, token efficiency, or leaderboard performance.

2. **No comparison to RLM-Qwen3-8B.** The natively recursive model of Zhang et al. (2026) is not in our comparison set.

3. **Synthetic fact identifiers.** The omitted-fact-safety metric relies on fixture-labeled identifiers; generalizing to natural-language settings requires a validated fact extractor with stable claim identifiers.

Ethics Statement

This work introduces a runtime contract for evidence-gated recursive language-model inference. It does not collect human data, fine-tune or release model weights, or propose a new long-context benchmark. The intended use is defensive: TG rejects parent claims that are unsupported by current-run child evidence, making unsupported model output detectable at a named runtime boundary rather than buried within a final answer. Two ethics-relevant caveats follow. First, TG's acceptance is a provenance property relative to a child evidence manifest, not a semantic truth claim; it reduces but does not eliminate the risk of unsupported generation reaching users. Second, the layered-memory observation (Section 4.6) shows that built-in agent-runtime memory can be injected into inner prompts in ways an operator may not realize; deployments should audit inner-prompt assembly when chaining memory layers. We do not anticipate that this runtime contract enables new offensive capabilities.

Reproducibility

An anonymized reproducibility archive is provided via the anonymous artifact link in the abstract. It contains committed fixtures, deterministic contract-test manifests, replay scripts, and table-generation metadata. For double-blind review, author-identifying repository URLs and branches are withheld until after the anonymity period. Live-provider experiments report provider family, run mode, and TG acceptance metadata without exposing author-identifying infrastructure.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Yuze He, Ji Qi, Lei Hou, Jie Tang, Yuxiao Dong, and Juanzi Li. 2024a. LongAlign: A recipe for long context alignment of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 1376–1395.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024b. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michał Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Bernd Bohnet, Vinh Q. Tran, Pat Verga, Roee Aharoni, Daniel Andor, Livio Baldini Soares, Massimiliano Ciaramita, Jacob Eisenstein, Kuzman Ganchev, Jonathan Herzig, Kai Hui, Tom Kwiatkowski, Ji Ma, Jianmo Ni, Lierni Sestorain Saralegui, Tal Schuster, William W. Cohen, Michael Collins, Dipanjan Das, et al. 2022. Attributed question answering: Evaluation and modeling for attributed large language models. *arXiv preprint arXiv:2212.08037*.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, et al. 2022. Improving language models by retrieving from trillions of tokens. In *Proceedings of the 39th International Conference on Machine Learning*.

Jifan Chen, Grace Kim, Aniruddh Sriram, Greg Durrett, and Eunsol Choi. 2024. Complex claim verification with evidence retrieved in the wild. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3569–3587.

I-Chun Chern, Steffi Chern, Shiqi Chen, Weizhe Yuan, Kehua Feng, Chunting Zhou, Junxian He, Graham

Neubig, and Pengfei Liu. 2023. Factool: Factuality detection in generative ai – a tool augmented framework for multi-task and multi-domain scenarios. *arXiv preprint arXiv:2307.13528*.

Yung-Sung Chuang, Benjamin Cohen-Wang, Zejiang Shen, Zhaofeng Wu, Hu Xu, Xi Victoria Lin, James R. Glass, Shang-Wen Li, and Wen-tau Yih. 2025. Selfcite: Self-supervised alignment for context attribution in large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 10839–10858.

Benjamin Cohen-Wang, Harshay Shah, Kristian Georgiev, and Aleksander Madry. 2024. Contextcite: Attributing model generation to context. In *Advances in Neural Information Processing Systems*.

Martin A. Conway and Christopher W. Pleydell-Pearce. 2000. The construction of autobiographical memories in the self-memory system. *Psychological Review*, 107(2):261–288.

Martin A. Conway, Jefferson A. Singer, and Angela Tagini. 2004. The self and autobiographical memory: Correspondence and coherence. *Social Cognition*, 22(5):491–529.

Luyu Gao, Zhuyun Dai, Panupong Pasupat, Anthony Chen, Arun Tejasvi Chaganty, Yicheng Fan, Vincent Zhao, Ni Lao, Hongrae Lee, Da-Cheng Juan, and Kelvin Guu. 2023a. RARR: Researching and revising what language models say, using language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16477–16508.

Tianyu Gao, Howard Yen, Jiatong Yu, and Danqi Chen. 2023b. Enabling large language models to generate text with citations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6465–6488.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, pages 79–90.

Hermes Agent. 2026. Hermes Agent: A self-improving AI agent runtime. Open-source repository. Repository authorship and URL omitted for anonymous review.

Chengyu Huang, Zeqiu Wu, Yushi Hu, and Wenya Wang. 2024. Training language models to generate text with citations via fine-grained rewards. *arXiv preprint arXiv:2402.04315*.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2023. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv preprint arXiv:2311.05232*.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. 2023. Atlas: Few-shot learning with retrieval augmented language models. *Journal of Machine Learning Research*, 24(251):1–43.

Adam Tauman Kalai and Santosh S. Vempala. 2023. Calibrated language models must hallucinate. *arXiv preprint arXiv:2311.14648*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. TruthfulQA: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Nelson F. Liu, Tianyi Zhang, and Percy Liang. 2023. Evaluating verifiability in generative search engines. *arXiv preprint arXiv:2304.09848*.

Jacob Menick, Maja Trebacz, Vladimir Mikulik, John Aslanides, Francis Song, Martin Chadwick, Mia Glaese, Susannah Young, Lucy Campbell-Gillingham, Geoffrey Irving, and Nat McAleese. 2022. Teaching language models to support answers with verified quotes. *arXiv preprint arXiv:2203.11147*.

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2023. FActScore: Fine-grained atomic evaluation of factual precision in long form text generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12076–12100.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Ouroboros. 2026. Ouroboros: Specification-First Workflow Engine for AI Coding Agents. Open-source repository. Repository authorship and URL omitted for anonymous review.

Fábio Perez and Ian Ribeiro. 2022. Ignore previous prompt: Attack techniques for language models. *arXiv preprint arXiv:2211.09527*.

Anirudh Phukan, Shwetha Somasundaram, Apoorv Saxena, Koustava Goswami, and Balaji Vasan Srinivasan. 2024. Peering into the mind of language models: An approach for attribution in contextual question answering. *arXiv preprint arXiv:2405.17980*.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. 2023. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*.

Jirui Qi, Gabriele Sarti, Raquel Fernández, and Arianna Bisazza. 2024. Model internals-based answer attribution for trustworthy retrieval-augmented generation. *arXiv preprint arXiv:2406.13663*.

Hannah Rashkin, Vitaly Nikolaev, Matthew Lamm, Lora Aroyo, Michael Collins, Dipanjan Das, Slav Petrov, Gaurav Singh Tomar, Iulia Turc, and David Reitter. 2023. Measuring attribution in natural language generation models. *Computational Linguistics*, 49(4):777–840.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*.

Weijia Shi, Xiaochuang Han, Mike Lewis, Yulia Tsvetkov, Luke Zettlemoyer, and Wen-tau Yih. 2024. Trusting your evidence: Hallucinate less with context-aware decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pages 783–791.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*.

Aviv Slobodkin, Eran Hirsch, Arie Cattan, Tal Schuster, and Ido Dagan. 2024. Attribute first, then generate: Locally-attributable grounded text generation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*.

James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018. FEVER: A large-scale dataset for fact extraction and VERification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 809–819.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Rose E. Wang, Pawan Wirawarn, Omar Khattab, Noah Goodman, and Dorottya Demszky. 2024a. Backtracing: Retrieving the cause of the query. In *Findings of the Association for Computational Linguistics: EACL 2024*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024b. Executable Code Actions Elicit Better LLM Agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 50208–50232.

Theodora Worledge, Judy Hanwen Shen, Nicole Meister, Caleb Winston, and Carlos Guestrin. 2023. Unifying corroborative and contributive attributions in large language models. *arXiv preprint arXiv:2311.12233*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.

Alex L. Zhang, Tim Kraska, and Omar Khattab. 2026. Recursive language models. *arXiv preprint arXiv:2512.24601*.

Jiajie Zhang, Yushi Bai, Xin Lv, Wanjun Gu, Danqing Liu, Minhao Zou, Shulin Cao, Lei Hou, Yuxiao Dong, Ling Feng, and Juanzi Li. 2024. Longcite: Enabling llms to generate fine-grained citations in long-context qa. *arXiv preprint arXiv:2409.02897*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *International Conference on Learning Representations*.

A Implementation Details

This appendix records implementation and reproducibility details useful for rerunning the artifact

but not needed to follow the main runtime-control argument. The main text uses conceptual names (controller, runtime adapter, current-run manifest, gate); this appendix records the concrete hooks.

Environment metadata. Table 5 records the live truncation artifact setup. These fields are not used as headline evidence for model quality, token efficiency, or monetary cost.

Table 5: Environment recorded for the live truncation artifact.

Field	Value
Hermes CLI	v0.11.0
Hermes endpoint	Custom OpenAI-compatible endpoint
Inner model	Provider-anonymized frontier-tier chat model
Ouroboros dependency	`5b4bc10e` claim-aware scorer commit
Evaluation command	`runtime-scaffold -truncation-benchmark`
Fixture	`rlm-long-context-truncation-v1`
Model calls	vanilla=1, recursive=5
Telemetry not persisted	latency, token count, monetary cost

Envelope and lifecycle fields. Each outer-to-inner handoff is rendered through an RA. The request envelope carries a mode, call-graph identifiers, recursion constraints, bounded context, and allowed evidence handles. The response carries a verdict, confidence, mode-specific result, evidence references, and residual gaps. The concrete lifecycle is: guardrail check, context binding and trace-start event, adapter invocation, envelope parsing and schema validation, evidence-reference admissibility check, atomic commit of accepted child results, and scheduling of decomposition, execution, summary, synthesis, retry, or termination.

Integration hooks. The dedicated recursive scaffold is the path exercised by the local `runtime-scaffold` entrypoint. The brownfield decomposition path enables the same RA-backed envelope only when the recursive mode is explicitly supplied; otherwise, the default LLM-only decomposition path remains unchanged. Environment-gated live tests are skipped unless the live-provider flag is set, so ordinary CI does not consume provider credits.

B Component Validation Matrix

Table 6 maps each component to the failure mode that would falsify it and to the experiment or replay that tests that failure mode.

C Atomic Task-Boundary Contract Test

Table 7 reports a 12-task deterministic fixture that instantiates the same boundary mechanism illustrated by Figure 2 (§3.6) in isolation from the larger 72-fixture contract ablation (Table 8). The broad single-shot condition commits all 12 injected unsupported parent claims because it lacks a child-evidence boundary. The one-level atomic and DFS conditions reduce unsupported commits to 0/12 under TG, while the typed semantic-miss column isolates the case for which SCG (Section 3.8) is required: a valid handle is not enough for typed semantic coverage, so the final row rejects typed claim terms absent from cited child evidence.

D Deterministic Contract Ablations

Table 8 reports deterministic contract ablations with no live provider or network calls. The scaled rows come from synthetic Qasper-style, HotpotQA-style, 2WikiMultihopQA-style, and ASQA-style rows and are intended as contract stress tests, not a live leaderboard.

E Post-Hoc Detection Vs. Pre-Commit Blocking

Table 9 operationalizes the distinction in Section 2.6 between diagnosing a committed answer and blocking the answer before commit. The post-hoc checker detects the unsupported bridge claim in every row, but because detection happens after answer commit, its pre-commit block rate is 0.000 and its unsupported-commit rate is 1.000. The corresponding EGRI condition sees the same unsupported attempt and blocks it before commit.

F Context-Budget Overhead

Table 10 reports the normalized call/token/latency cost of the four context-budget conditions (Section 4.5) for the same 10 deterministic long-dossier fixtures. Units are normalized artifact units, not provider billing.

G Vanilla Live Comparison

Table 11 reports the side-by-side live comparison on `rlm-long-context-truncation-v1`. Both runs use the same `HermesCliRuntime`, fixture, and scoring code.

H Reproduction Commands

The live truncation artifact can be reproduced with one command.

```
runtime-scaffold --truncation-benchmark
```

Table 6: Component validation matrix. Table 1 defines the architecture contract; this table states which falsifier is exercised by which evidence tier and what pass signal is reported.

Component	Falsifier exercised	Evidence tier	Artifact	Pass signal
RC	Child/model self-invocation, bounds bypass, or state mutation outside the controller.	live + replay	Appendix A; Table 2	24/24 live cells complete through the RC-owned lifecycle; persisted replay and in-process gate checks show validation failures do not mutate parent state.
RA	Adapter-specific control drift or hidden recursive ownership inside an adapter.	live	Table 2	Same contract and fixtures pass across 3 adapter families, 24/24 cells.
TE	Schema-invalid handoff or child I/O mutating state before validation.	lifecycle + replay	Appendix A	Every child output in the reported fixtures is parsed and schema-validated before call-graph, artifact, or parent-state mutation.
CEM	Omitted, rejected, or memory-only facts entering current-run evidence state.	replay + det. + live	Tables 3, 4, 16	Gated rows commit no omitted or memory-only facts; memory-enabled live cells report 0 memory-only factual commits.
TG	Unsupported parent claim committed before a manifest-membership check.	replay + det. + live smoke	Tables 3, 8, 15	8/8 unsafe primary replays blocked; 0/72 contract-trap commits; in-process rejects fire before CLI success.
SCG	Valid evidence handle cited while unsupported typed terms are asserted.	det.	Table 7	Typed-semantic-miss commits fall from 12/12 without SCG to 0/12 with SCG.
GOM	Durable memory supports a committed factual parent claim.	live + det.	Table 16; Appendix L	24/24 memory-enabled live cells pass with 0 memory-only factual commits; deterministic memory controls keep memory-answer acceptance at 0.

Table 7: Atomic task-boundary contract test over 12 deterministic synthetic tasks. Values are bad-event counts.

Condition	Atomic leaves?	Unsupported commits ↓	Typed-semantic-miss commits ↓	Max depth
Broad single-shot	no	12/12	not tested	0
One-level atomic + TG	yes	0/12	12/12	1
DFS + TG	yes	0/12	12/12	2
DFS + TG + SCG	yes	0/12	0/12	2

Table 8: Deterministic contract ablations. No row uses live provider or network calls.

Condition	Fixture set	Evaluations	Unsupported commits ↓
EGRI	contract traps	72	0/72 = 0.000
EGRI substrate w/o TG	contract traps	72	72/72 = 1.000
Scaled fixture, gated policies	synthetic family-style	160	0/160 = 0.000
Scaled fixture, ungated policies	synthetic family-style	240	160/240 = 0.667

Table 9: Post-hoc detection versus pre-commit blocking over 80 deterministic examples. Detection after commit can diagnose the unsupported claim but cannot prevent the parent state mutation.

Condition	Detects unsupported claim ↑	Blocks before commit ↑	Unsupported commits ↓
Post-hoc checker	80/80 = 1.000	0/80 = 0.000	80/80 = 1.000
TG pre-commit gate	N/A	80/80 = 1.000	0/80 = 0.000

Table 10: Deterministic context-budget overhead. Units are normalized artifact units, not provider billing. Recursive conditions spend additional bounded calls and token units for inspectable evidence bandwidth; TG adds the pre-commit block without changing the recursive overhead.

Condition	Fixtures	Model calls	Token units	Latency units	Unsupported-commit rate ↓
Single-shot context window	10	1	1,680	2.000	1.000
Flat top-k retrieval	10	1	1,712	2.000	1.000
Recursive evidence cache	10	8	2,784	9.667	1.000
DFS + TG	10	8	2,784	9.667	0.000

Without an API key, the persisted artifact can be replayed.

```
python -m egri.replay \
    benchmarks/rlm-long-context-truncation-v1.
        json
```

Both commands print the headline metrics from Table 11.

The TG enforcement demo runs without live model calls.

```
python3 scripts/run-traceguard-demo.py
```

It accepts a safe parent synthesis, rejects an omitted-fact claim, and rejects chunk-only evidence that does not identify supported fact IDs.

The 24-cell live primary portability matrix can be rerun with provider credentials configured.

```
HERMES_INFERENCE_PROVIDER=<provider> \
PROVIDER_API_KEY=<redacted-or-env-configured> \
uv run --extra dev --extra live \
  python scripts/run-live-portability-matrix.py \
  --mode live-primary --fixtures 8 --families
      all \
  --timeout-seconds 240 \
  --output-prefix live-portability-primary
```

The memory-enabled primary run uses the same matrix with the memory store turned on.

```
uv run --extra dev --extra live \
  python scripts/run-live-portability-matrix.py \

  --mode live-primary --fixtures 8 --families
      all \
  --timeout-seconds 240 \
  --memory-mode read-write \
  --memory-store experiments/live-portability-
      memory-primary-fixed.jsonl \
  --output-prefix live-portability-primary-
      memory-fixed
```

Table 11: Side-by-side comparison on `rlm-long-context-truncation-v1`.

Metric	Vanilla single call	EGRI recursive run
Inner model calls	1	5
Answer quality score	1.00	1.00
Omitted-fact safety	1.00	1.00
Claimed omitted facts	∅	∅
Cited retained facts	LC-001–LC-004	LC-001–LC-004
Current-run child manifest	N/A	present
Parent synthesis observed	N/A	true
TG accepted parent	N/A	true

We additionally include a model-free scorer micro-suite.

```
python3 scripts/run-claim-aware-suite.py
```

The current suite contains seven controlled completion shapes, and all seven pass under the corrected scorer. The cases cover guarded omitted-ID mentions, positive omitted-fact assertions, omitted evidence references, chunk-only citations without fact evidence, missing truncation-boundary reports, and missing retained facts.

We also include a generated synthetic scorer suite.

```
python3 scripts/run-synthetic-omitted-fact-
    benchmark.py
```

It creates 108 truncation fixtures by varying the fact count, the retained/omitted ratio, the distractor density, and the omitted-claim target. Each fixture is scored against seven deterministic completion strategies, yielding 756 sanity checks for scorers. All pass in the committed artifact. This scorer suite validates the measurement harness, not live-model quality.

Finally, we include an ablation study for the unsupported-commit-rate contract.

```
python3 scripts/run-unsupported-claim-rate-
    benchmark.py
```

This contract-ablation script generates 72 fixtures and evaluates 6 execution contracts. They are loose single call, guarded single call, chunk-only map-reduce, leaky map-reduce, EGRI, and EGRI substrate w/o TG. The EGRI contract has an unsupported-commit rate of 0.000. The EGRI substrate w/o TG contract has an unsupported-commit rate of 1.000. The result is deterministic and should be read as a contract ablation, not as a live-model comparison.

The memory-runtime benefit control test is also deterministic and model-free.

```
python3 scripts/run-memory-runtime-benefit-
    benchmark.py
```

The three optional-memory control tests are generated separately.

```
python3 scripts/run-memory-contribution-
    benchmarks.py
```

It generates 20 fixtures, 4 provider-failure profiles, and 3 memory policies for 240 evaluations. The GOM policy improves initial TG acceptance from 0.2500 to 1.0000 relative to the no-memory policy, reduces the mean number of repair calls from 0.2500 to 0.0000, and keeps the final unsupported-commit rate at 0.0000. This is a runtime-control result: it proves that a schema/retry memory prior can remove known repair work in the contract, not that live models become more accurate.

The optional-memory script writes three additional artifacts: `memory-contamination`, `layered-memory-ablation`, and `adaptive-repair-memory`. Together, they support a bounded framing: EGRI can use runtime memory as policy while preventing memory from becoming evidence unless a current child trace establishes it.

The context-budget pressure stress test is generated by the artifact module and ships as a paper-facing artifact.

```
uv run python -m egri.benchmarks.context_budget
    \
  --output-dir experiments/context-budget-
      pressure
```

It writes 40 row-level deterministic evaluations, a CSV summary, and metadata with explicit `live_model_calls=0` and `network_calls=0` safety flags. The artifact is intended for the context-budget table, not as a live model-quality result.

The additional deterministic artifact package generates the scaled multidataset fixture, the post hoc checker comparison, and the normalized overhead table.

```
uv run python -m egri.benchmarks.
    emnlp_additional \
  --repo-root /path/to/egri-artifact \
  --output-dir experiments/emnlp-additional \
  --examples-per-dataset 20
```

It writes 80 synthetic family-style fixture rows, a 400-output offline contract-suite run, an 80-example post-hoc vs pre-commit comparison, and a four-row overhead CSV. The meta-

data again records `live_model_calls=0` and `network_calls=0`; the result is deterministic runtime-control evidence, not a live-provider quality benchmark.

I Complete Experiment Inventory

Table 12 lists the experiment families and their evidential scope. We intentionally keep live-provider evaluations, deterministic contract tests, and offline fixture artifacts in the same inventory, but mark their evidence type separately. This makes the evidential boundary explicit: live matrices support portability of the runtime contract, deterministic suites support TG and memory-policy mechanisms, and the offline multidataset mini-suite supports harness readiness rather than final external validity claims.

Table 12: Experiment inventory and evidential scope, part 1. “Live” rows used real agent/provider execution; “det.” rows are deterministic contract or fixture artifacts.

Experiment family	Evidence type	Scale	Supports	Main caveat
Long-context truncation fixture	live + replay	vanilla 1 call vs recursive 5 calls	Recursive execution exposes child evidence and parent synthesis while corrected scoring shows parity	Not a quality win over vanilla on this fixture
Primary portability matrix	live	8 fixtures $\times$ 3 runtime families = 24 cells	Same TG contract passes across Hermes+GLM, Claude Code, and Codex	Does not establish cost, latency, or token superiority
Primary portability with memory	live	8 fixtures $\times$ 3 runtime families = 24 cells	Operational memory priors coexist with current-run TG acceptance	Memory is not factual evidence and does not prove answer-quality improvement
Unsupported-commit / TG ablations	det.	contract fixtures and parent-claim variants	Unsupported parent claims can be rejected before commit	Handle-level admissibility is simpler than semantic attribution
Memory-runtime benefit	det.	20 fixtures, 3 policies, 240 evaluations	Guarded operational memory (GOM) reduces schema/repair work and rejects answer-memory contamination	Runtime-control result, not live model-quality result
Memory contamination robustness	det.	12 fixtures, 6 policies, 72 evaluations	TG rejects adversarial memory-answer facts absent from current-run child evidence	Not a general prompt-injection defense

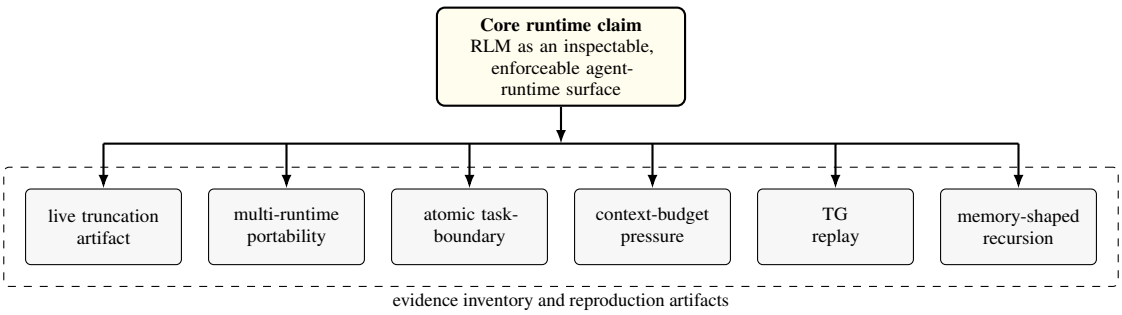

Figure A1: Evidence map for the appendix inventory. The inventory is organized as supporting artifacts for one system’s claim rather than as independent leaderboard claims: EGRI exposes recursive execution as a runtime surface whose child evidence, parent synthesis, replay, memory priors, and failure gates can be inspected and enforced.

Table 13: Experiment inventory and evidential scope, part 2. The inventory is intentionally inclusive to separate live portability evidence, deterministic contract tests, and offline fixture artifacts.

Experiment family	Evidence type	Scale	Supports	Main caveat
Atomic task-boundary fixture	det.	12 fixtures $\times$ 4 traversal/contract conditions = 48 evaluations	DFS reaches atomic, contract-checkable leaves; TG rejects unsupported parent facts and SCG rejects explicit typed semantic misses before commit	Does not make child generation deterministic or solve open-world semantic evaluation
Layered memory 2$\times$2 ablation	det.	12 fixtures $\times$ 4 provider profiles $\times$ 4 policies = 192 evaluations	runtime prompt memory and GOM repair different failure classes	Simulated provider profiles, not live provider calls
Adaptive repair memory	det.	16 evaluations, 2 policies	Reused missing-handle priors reduce repeated repair calls while preserving final acceptance	Narrow failure family
Context-budget pressure	det.	10 fixtures $\times$ 4 conditions = 40 evaluations	Recursive evidence caching raises supported-fact recall from 0.375 for single-shot and 0.625 for flat top-k to 1.000 under a fixed parent budget; TG reduces unsupported commits from 1.000 to 0.000	Synthetic pressure fixture, not live model quality
Deterministic scale-up	det. offline	80 synthetic family-style examples $\times$ 5 policies = 400 outputs	Exercises the multidataset materialization path at larger scale; gated policies retain unsupported-claim rate 0.0000	Deterministic fixture, not official sampled leaderboard data
Post-hoc vs pre-commit gate	det.	80 examples $\times$ 2 conditions	Post-hoc checker detects unsupported claims but cannot block commit; TG blocks before parent state mutation	Mechanistic baseline, not a probabilistic attribution model
Cost/token/latency overhead	det.	4 context-budget conditions	Reports explicit call, pseudo-token, and latency-unit tradeoff for evidence bandwidth and zero unsupported commits	Normalized units, not provider billing
In-process `runtime-scaffold` gate replay	det. + live smoke	persisted replay plus local wrapper smoke	Gate rejects nonexistent child handles and memory-answer injection before CLI success	Repository path proof, not all upstream execution paths
Real multidataset mini-suite artifacts	det. offline	4 dataset families, 8 examples, 5 policies, 5 ablations, 10 table rows	Harness can materialize fixtures, run offline contract tests, and generate paper tables	Tiny CI-scale fixture, retained only as smoke test

J Portability Fixture Classes

Table 14 lists the eight live-portability fixture classes used in Tables 2 and 16. Each fixture has six canonical facts: four retained facts that may enter the CEM and two omitted facts that must not support parent claims. The pass criterion is identical for every row: the parent synthesis must be schema-valid, all committed claims must cite retained `fact_id`/`evidence_handle` pairs from the CEM, omitted facts must not be committed, and chunk-only or memory-only references must be rejected. These are not intended as natural-language QA examples with hidden gold answers; they are contract fixtures. Their purpose is to expose the difference between (i) a baseline LLM or EGRI substrate w/o TG policy that may produce a plausible free-form answer, (ii) a post-hoc checker that can flag an already committed unsupported claim, and (iii) EGRI, which blocks the unsupported parent proposal before commit. The final column makes explicit which component invariant each fixture stresses, so the suite is a contract-validation map rather than an arbitrary mini-suite. The deterministic baseline tables in the main text instantiate those comparisons on a larger scale.

K In-Process Gate Implementation Details

The first paper replay used the dependency recursive-scaffold path to decompose a mechanically extracted paper target into four child model calls and one parent synthesis call. A separate post-run script then normalized the persisted parent into TG's fact/evidence-handle form. That proved compatibility, but it did not prove that the CLI itself enforced the gate.

EGRI now adds a project-local `runtime-scaffold` console script wrapper. The wrapper installs a process-local patch around the upstream recursive-scaffold helper and then delegates to the upstream Typer app. The patch does not mutate the dependency package. After parent synthesis, it builds the CEM from the current run's child results: `child_result_id` plus child chunk handle becomes a TG evidence item. Parent `supported_by_child_result_ids` entries become TG claim references. Any parent fact that cites a missing child result, wrong chunk handle, or memory-only answer remains outside the CEM and is rejected.

We also executed the live wrapper path after installing the gate, using `uv run` with the project-local `runtime-scaffold` entrypoint over the paper key-section target (Table 15, last row). The run completed five atomic execution sub-calls, generated an RLM tree of depth 1, and printed a TG accept with an unsupported rate of 0.0000 and 4 accepted claims before reporting command success. This establishes that the gate is no longer merely a post-run verifier for this repository path; it is applied in-process before the local `runtime-scaffold` wrapper reports success.

L Deterministic Memory Control Tests

This appendix contains the deterministic, model-free runtime-control tables and prose referenced from Section 4.6.

Memory-as-guidance background. EGRI's memory boundary draws on autobiographical memory theory: Conway and Pleydell-Pearce's self-memory system separates experience-near episodic records from a goal-directed *working self* (Conway and Pleydell-Pearce, 2000), and later work frames remembering as balancing *correspondence* with experience and *coherence* with current goals (Conway et al., 2004). We borrow only the design separation between goal-directed retrieval control and experience-grounded correspondence; we do not model a working self, autobiographical knowledge, or human recollection.

Implementation prompt memory. The integration revealed a second, emergent memory layer in the concrete RA implementation: prompt memory can be injected into inner turns before the explicit child or parent JSON envelope. We treat this as a behavioral prior, not as part of the evidence contract. In a real-document replay over the README and this paper's memory section, parent synthesis correctly reported that the document evidence supports GOM, while the locally observed implementation prompt-memory layer remains a separate observation rather than a document-supported claim.

Memory-runtime benefit (Table 17). The control test is generated without live model calls and compares three memory policies across generated fixtures and provider-failure profiles. The GOM policy models a schema/retry prior that tells the parent to preserve `fact_id` and `evidence_handle` together, eliminating the repair call required by the missing-handle-prone no-memory policy. The

Table 14: Eight live-portability fixture classes and the component invariants they stress. Each fixture is crossed with three runtime families, producing the 24 cells in Table 2 and reused in Table 16.

Fixture id	Category	Retained evidence role	Unsafe parent trap	Component/invariant stressed
`simple-truncation-01`	simple truncation	retained chunks define the CEM	hidden appendix/deprecated scorer facts	CEM current-run evidence only; TG membership
`simple-truncation-02`	simple truncation	retained chunks define child evidence	parent imports facts from omitted chunks	RC/TE bounded context; TG membership
`distractor-heavy-01`	distractor-heavy	retained facts survive distractor padding	distractors become claims about recursion wins	TE context bounds; SCG typed terms
`distractor-heavy-02`	distractor-heavy	retained operational facts only	metadata or credential-like notes committed	TG rejects omitted/non-evidence claims
`cross-chunk-dependency-01`	cross-chunk dependency	both retained endpoints must be cited	dependency claim cites only one endpoint	CEM fact/handle pairs; SCG term coverage
`cross-chunk-dependency-02`	cross-chunk dependency	shared fixture JSON supports parity	adapter-specific claim drift	RA portability under same TG logic
`omitted-fact-temptation-01`	omitted-fact temptation	retained evidence excludes quality-win facts	omitted benchmark-win claim committed	TG pre-commit omitted-fact rejection
`chunk-only-citation-trap-01`	chunk-only citation trap	retained chunk exists with fact handles	citation lacks `fact_id`	TG requires fact ID plus evidence handle

Table 15: In-process TG gate behavior before the local `runtime-scaffold` wrapper reports CLI success. Counts are unsupported parent proposals over structured parent references.

Case	Unsupported proposals	Expected outcome	Observed outcome	Reason
Missing child-evidence handle	1/23 = 0.0435	reject	reject	cited handle absent from the CEM
Handle repaired from current trace	0/23 = 0.0000	accept	accept	repaired handle found in the CEM
Memory-only answer injected	1/20 = 0.0500	reject	reject	fact absent from the CEM
Live wrapper smoke (`uv run runtime-scaffold`)	0/4 = 0.0000	accept	accept	all claims manifest-backed

Table 16: Optional memory-enabled live portability over the same eight fixture classes; the pass criterion of Table 2 extends with one additional bad-event constraint: zero memory-only factual commits.

Family	Memory mode	Fixtures passed	Memory-only factual commits ↓
`hermes_glm`	`read-write`	8/8	0
`claude_code_opus47`	`read-write`	8/8	0
`codex_gpt55`	`read-write`	8/8	0

Table 17: Deterministic memory-runtime benefit control test. The benefit is runtime-control performance: fewer initial rejections and repair calls, not live model quality.

Policy	Initial accept	Final accept	Mean repairs	Mean calls
`no_memory`	0.2500	0.5000	0.2500	1.2500
`guarded_operational_memory_prior`	1.0000	1.0000	0.0000	1.0000
`unsafe_answer_memory_prior`	0.0000	0.0000	0.0000	1.0000

Table 18: Deterministic optional-memory control tests. All results are model-free runtime-control tests, not live quality benchmarks.

Control test	Initial accept	Mean repairs	Memory answer accepted
Contamination, unguarded adversarial	1.0000	0.0000	1.0000
Contamination, TG adversarial	0.0000	0.0000	0.0000
Layered memory, both layers on	1.0000	0.0000	0.0000
Adaptive repair memory	0.8750	0.1250	0.0000

unsafe answer-memory policy models contaminated memory that adds a factual claim; TG rejects it. This supports only a runtime-performance claim: GOM can reduce validation and repair work for known schema failures while preserving the current-run evidence boundary.

Three optional-memory control tests (Table 18). First, the contamination robustness control test adds an unsupported answer fact from persistent memory; without TG, the adversarial-memory policy accepts that memory answer in every case, but with TG, the accepted memory-answer rate is 0.0000 because the fact is absent from the current-run accepted child evidence manifest. Second, the layered-memory ablation evaluates a 2×2 design (runtime prompt memory on/off and GOM on/off): runtime prompt memory models a formatting prior that prevents chunk-only parent synthesis, while GOM models an evidence-handle prior that prevents missing handles, and the both-layers-on cell reaches 1.0000 initial acceptance with zero repair calls (the remaining three cells are in the artifact log produced by `run-memory-contribution-benchmarks.py`). Third, the adaptive repair memory control test records a missing-handle prior after the first repair and reuses it on later related tasks; mean repair calls fall to 0.1250 (vs the no-prior baseline in the artifact log) while the memory-answer acceptance rate stays at 0.0000.

M Extended Related Work

This appendix expands the five thematic clusters surveyed in Section 2, elaborates the per-comparator distinctions summarized in Section 2.6, and presents the positioning summary matrix referenced from that section.

Recursive inference and executable agent runtimes. Zhang et al. (2026) introduce RLM as an inference strategy in which an outer model recursively invokes bounded sub-calls over transformed slices of long input. We use that work as the direct foundation, but change the deployment unit: EGRI does not post-train a recursive checkpoint or require a special Python REPL environment. Instead, it lifts the same control surface into an existing agent runtime that already knows how to call models, tools, and child tasks. This places the system in the realm of execution-oriented agent work, such as CodeAct (Wang et al., 2024b), where model outputs become actions within an execution substrate. The distinction is that EGRI treats recursive sub-calls and parent synthesis as a typed, replayable runtime contract rather than as incidental use of tools. Hermes Agent (Hermes Agent,

Table 19: Positioning of EGRI against the closest lines of prior work. "No model change" = applies without retraining the inner model; "Pre-commit gate" = admissibility is checked before state mutation; "Deterministic" = the admissibility relation is a typed set-membership check, not a probabilistic score; "Runtime contract" = the technique is a typed pre-commit admissibility check at a named synthesis boundary.

Approach	No model change	Pre-commit gate	Deterministic	Runtime contract
RLM-Qwen3-8B (Zhang et al., 2026)	✗	✗	✗	✗
Training-time citation grounding (Nakano et al., 2021; Menick et al., 2022; Gao et al., 2023b)	✗	✗	✗	✗
Post-hoc attribution (Cohen-Wang et al., 2024; Chuang et al., 2025)	✓	✗	✗	✗
Post-hoc factuality / revision (Gao et al., 2023a; Min et al., 2023; Chern et al., 2023)	✓	✗	✗	✗
Executable-action agents (Wang et al., 2024b)	✓	✗	✗	partial
EGRI (ours)	✓	✓	✓	✓

2026) and Ouroboros (Ouroboros, 2026) therefore matter as infrastructure: Hermes provides bounded callable inner execution, while Ouroboros owns state, scheduling, trace replay, and rejection before commit.

Long-context decomposition and recursive evidence. A large body of work studies how LLMs fail or degrade on long inputs. The "lost in the middle" effect shows that models can underuse information placed away from the beginning or end of context (Liu et al., 2024), and LongBench provides broad evaluation tasks for long-context understanding (Bai et al., 2024b). LongCite focuses the same pressure on long-context QA with fine-grained citations (Zhang et al., 2024). EGRI is not another long-context benchmark and does not claim broad superiority in quality from context partitioning alone. Its narrower claim is that partitioned sub-calls become operationally useful when their outputs are preserved as inspectable child evidence and when parent synthesis can be blocked if it cites evidence that no child produced.

Evidence-grounded generation and citation-producing systems. Several systems train or prompt models to produce sourced answers, including WebGPT (Nakano et al., 2021), verified-quote supervision (Menick et al., 2022), and citation-generating LLMs (Gao et al., 2023b). Other work makes grounded generation more local or structured, including Attributed Question Answering (Bohnet et al., 2022) and attribute-first grounded text generation (Slobodkin et al., 2024). These systems motivate the same user-facing requirement as TG: a final answer should expose where its claims come from. The enforcement boundary is different. EGRI does not accept a parent answer merely because it emitted citation-looking text. TG checks whether each structured parent claim cites a fact and evidence handle that exists in the current-run accepted child evidence manifest.

Attribution, context contribution, and verifiability. Context attribution work asks whether a generated statement can be tied back to the context that caused or supports it. ContextCite estimates which context fragments contributed to a generation through ablation and sparse surrogate modeling (Cohen-Wang et al., 2024); SelfCite uses context ablation to score citation necessity and sufficiency, then improves citation quality with best-of-N sampling and preference optimization (Chuang et al., 2025). Other work studies answer attribution

for RAG from model internals (Qi et al., 2024), distinguishes corroborative from contributive attribution (Worledge et al., 2023), and backtraces the causes of queries or outputs (Wang et al., 2024a). Evaluation work on NLG attribution and verifiability makes clear that citations are only useful when they support checking (Rashkin et al., 2023; Liu et al., 2023). EGRI takes a simpler, more mechanical path than semantic attribution: it turns admissibility into a deterministic runtime contract on explicit child handles. If a fact is absent from the accepted child trace, the parent cannot commit it.

Factuality, repair, and memory contamination. Post-hoc factuality and revision systems such as RARR (Gao et al., 2023a), FActScore (Min et al., 2023), FacTool (Chern et al., 2023), and complex claim verification with retrieved evidence (Chen et al., 2024) address the broader problem of unsupported generation. Hallucination surveys and theory further motivate why factual evidence cannot be assumed from the model distribution alone (Huang et al., 2023; Kalai and Vempala, 2023). EGRI uses that lesson at the runtime boundary: memory may guide decomposition, routing, or repair, but it is not admissible factual evidence. This boundary also connects to prompt- and context-contamination work, including indirect prompt injection (Greshake et al., 2023) and early prompt-injection attack taxonomies (Perez and Ribeiro, 2022). Long-context alignment and decoding work such as LongAlign (Bai et al., 2024a), context-aware decoding (Shi et al., 2024), fine-grained citation rewards (Huang et al., 2024), and contextual QA attribution (Phukan et al., 2024) are complementary: they can improve model behavior, while TG constrains what the runtime is allowed to accept.

Distinction from RLM (Zhang et al.). Zhang et al. (2026) operationalize recursion by post-training a dedicated recursive checkpoint (RLM-Qwen3-8B) inside a Python REPL environment. EGRI takes the opposite path. The recursive control surface is lifted into an existing agent runtime via an RA/TE boundary; the inner model is swappable and need not be post-trained. We do not claim to match RLM-Qwen3-8B on quality (see the Limitations). Our contribution is that the same style of runtime control can be exposed without a new checkpoint.

Distinction from training-time citation grounding. WebGPT (Nakano et al., 2021), verified-quote supervision (Menick et al., 2022), inline-citation LLMs (Gao et al., 2023b), AQA (Bohnet et al., 2022), and attribute-first grounded generation (Slobodkin et al., 2024) all modify the model so that the generated text emits citation-looking tokens. Admissibility is implicit in the trained policy. EGRI leaves the inner model untouched and instead validates a structured parent-synthesis object against the current-run-accepted child evidence manifest. Even if the inner model emits no citations, TG can still admit or reject the parent.

Distinction from post-hoc attribution. ContextCite (Cohen-Wang et al., 2024) and SelfCite (Chuang et al., 2025) score how strongly a generated statement is supported by retrieved context. The score is probabilistic and lives downstream of generation. EGRI checks a deterministic relation: each structured fact claim in the parent synthesis must name a `fact_id/evidence_handle` pair that exists in the manifest assembled from accepted child outputs. Missing pairs are rejected before state mutation, not flagged after the fact.

Distinction from post-hoc factuality and revision. RARR (Gao et al., 2023a) retrieves web evidence and rewrites the generated answer; FActScore (Min et al., 2023) and FacTool (Chern et al., 2023) score factual atoms after the answer is produced. EGRI blocks the commit at the parent-synthesis boundary, so an unsupported claim never becomes an accepted state in the first place. The contracts are complementary: RARR/FActScore/-FacTool operate on a committed answer, while TG operates on a not-yet-committed parent synthesis.

Distinction from executable-action agents. CodeAct (Wang et al., 2024b) treats model outputs as actions inside an execution substrate, which yields a runtime contract on what an agent may *execute*. EGRI adds a typed contract on what a parent may *claim* from its child evidence. The two contracts compose: an execution agent can adopt EGRI's parent-synthesis gate as the admissibility check before mutating its state.

N Future Work: Optional Operational Memory

This appendix expands the future-direction sentence in Section 5.

EGRI does not train the underlying model during RLM execution. The implemented memory path targets a different opportunity: runtime memory, skills, and traces can give the recursive scaffold a persistent operational memory. The current implementation stores a narrow subset of provider-specific schema and repair observations. Future work should broaden that policy to prior decompositions, routing decisions, retry strategies, and task-family specialization.

Under the same memory-is-not-evidence boundary established in Section 5, memory can shape the policy for the next run without supporting any factual parent claim. It can influence how to decompose a task family, which provider to route extraction or synthesis to, which schema to require, and which retry prompt to use after a guardrail rejection. The implemented run already recalls provider schema stability as a policy prior. A future version should recall missing-handle rejections as schema-repair priors, retry child calls when handles are null, and learn provider-routing policies from repeated TG outcomes. The repaired parent would still need to cite current-run evidence from the new trace before TG accepts it.

This direction is useful only if memory is curated. Raw progress logs, provider chatter, and fixture answers should not become durable recall. The memory record should be an abstract operational signal, such as "this provider is slow but usually schema-stable" or "this task family needs fact-level handles rather than chunk-only handles." Fixture use also requires contamination controls: no-memory baselines, answer-memory disabled, fixture content excluded from durable memory, and separate trace-memory policy runs. The current implementation enforces that separation through an allowlisted memory schema and a current-run-accepted child evidence manifest.