EVITRACE-DE: EVIDENCE-GATED GENERATION OF TRACEABLE AND EXECUTABLE DETECTION RULES FROM CTI WITH LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Detection engineering converts attack behaviors described in cyber threat intelligence (CTI) into executable detection rules over security telemetry, but it remains expert-intensive and manual. Tool-use LLM agents can retrieve reports, inspect schemas, and query telemetry, but retrieval alone does not prevent unsupported conditions or event relations from entering the final rule, producing false or unexecutable rules. We study this failure mode as the evidence-commit gap and present EVITRACE-DE, an evidence-gated framework that separates open-ended LLM reasoning from rule construction through explicit evidence verification, enabling accurate and traceable rule generation. An LLM agent proposes candidate predicates and event relations, and programmatic validators verify candidate evidence and close the corresponding behavioral, schema, telemetry, or relational preconditions. Only candidates that pass all required verifications are admitted to P-DIR, an intermediate representation backed by evidence that binds each admitted condition to its supporting evidence. Deterministic compilers then generate KQL and Sigma rules using only the predicates and event relations admitted into P-DIR. The evidence-verification state also provides each LLM call with only the context needed for the current verification step, avoiding repeated full-history prompts. In CTI-REALM-50, EVITRACE-DE improves the normalized score from 0.537 to 0.752 and KQL F1 from 0.496 to 0.804, achieves a valid-submission rate of 100% and reduces the mean token usage per task by 98.7% compared to ReAct. In an independent AWS CTI-to-Sigma task, adapting verification to threat reports and API field definitions improves API-call F1 by 35.9% over a ReAct-style agent using the same model.

1 INTRODUCTION

Cyber threat intelligence (CTI) describes adversary behavior, yet operational use requires executable detection rules over security telemetry to detect and respond to intrusions. Such telemetry consists of security-relevant event records from endpoints, cloud services, and container platforms, and audit logs are a common type. A detection rule filters these records through predicates defined over a data source, field, operator, and value, and may further relate multiple records. Translating CTI into platform-specific conditions therefore depends on expert knowledge of schemas, operation semantics, and observed telemetry, and it often requires substantial manual effort (Chakraborty et al., 2026; Meng et al., 2026).

Considerable progress has been made in automating the generation of detection rules, with prior work extracting structured behavior from CTI or generating Sigma and related rules (Büchel et al., 2025; Li et al., 2022; Cheng et al., 2025; Cai et al., 2026; Schwartz et al., 2025), while tool-use agents retrieve intelligence, inspect schemas and query telemetry during generation (Yao et al., 2023; Schick et al., 2023; Lu et al., 2024; Chakraborty et al., 2026). However, the information retrieved by such tools typically remains confined to the model context, and its verification status does not govern whether the model may encode the corresponding value or event relation in the final rule.

For example, consider a Kubernetes client-authentication Certificate Signing Request (CSR). CTI may describe an HTTP `POST` to the CSR collection endpoint, whereas the audit field

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

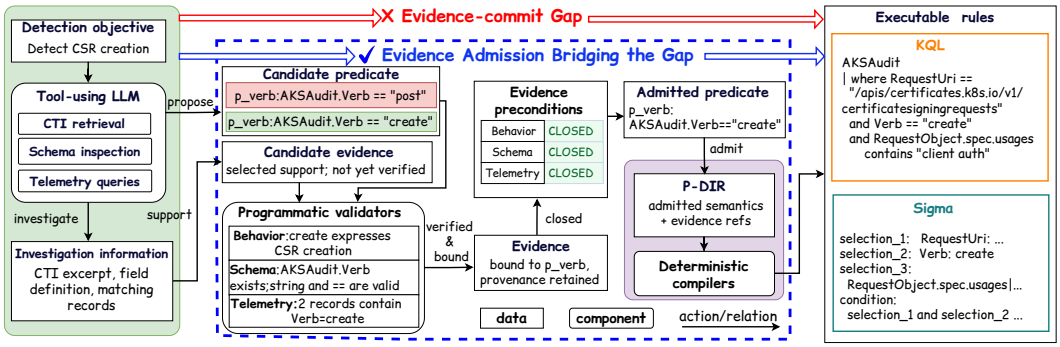

Figure 1: The evidence-commit gap and evidence-gated generation for one CSR predicate.

`AKSAudit.Verb` records the resource operation as `create` (Kubernetes Authors, 2026; Microsoft, 2026). Even after retrieving this mapping, an agent may still emit `Verb=post`, copy an instance-specific identifier, omit a necessary condition, or correlate separately observed stages without relational support. We refer to this disconnect between available investigation information and rule writing, left unconstrained by verification status, as the **evidence-commit gap**. Appendix A specifies the attacker, CTI analyst, defender, and trust assumptions under which we study this gap.

The gap harms both correctness and traceability. A tool trace shows what an agent inspected, but it does not bind each final condition to the evidence that supports it, forcing analysts to reconstruct the provenance from a long trajectory (Valletta et al., 2026). The complete history prompt also repeatedly reintroduces completed investigations, which increases the cost (Jiang et al., 2026; Wang et al., 2025b). We therefore require a model-external boundary through which verified evidence can control semantic writes.

We present EVITRACE-DE to close this gap by **evidence admission**. A predicate is an atomic condition over a single event, such as `AKSAudit.Verb=="create"`, while an **event relation** connects events through a subject, join key, or temporal order. The LLM agent proposes candidate predicates and, when necessary, candidate event relations and selects **candidate evidence** from the information retrieved by the tools. Programmatic validators accept candidate evidence only after verifying its content, binding it to the predicate or relation it supports, and retaining its provenance. The **Evidence State** records this evidence together with each candidate's **evidence preconditions** for behavioral, schema, telemetry, or relational evidence, and a successful verification closes the corresponding precondition. Only after the validators close every required precondition does evidence admission allow a candidate into the provenance-bound detection intermediate representation (P-DIR). By this process, the LLM agent proposes and revises, programmatic admission controls write eligibility, and deterministic compilers render only the semantics admitted into P-DIR.

Figure 1 instantiates these terms for the CSR predicate, where the red path generates detection rules directly from context, while the controlled path verifies and binds candidate evidence, closes preconditions, admits `Verb=create` into P-DIR, and deterministically compiles KQL and Sigma.

We evaluated EVITRACE-DE on 50 CTI-REALM tasks and an independent AWS CTI-to-Sigma setting. EVITRACE-DE raises KQL F1 from 0.496 to 0.804, achieves a valid-submission rate of 100%, reduces the mean use of tokens by 98.7% relative to ReAct, and improves AWS API-call F1 by 19.7 percentage points. We make three contributions:

1. We identify the **evidence-commit gap** and separate open-ended model proposal from programmatic evidence verification and write eligibility, preventing unsupported semantics from entering executable rules.
2. We introduce P-DIR as a traceable admitted-semantic boundary and deterministically compile its predicates and relations into KQL and Sigma, enabling condition-level provenance and reliable rule reproduction.
3. Experiments across models, mechanisms, and two rule-generation settings show higher rule quality, complete evidence paths for admitted predicates, and lower context cost.

2 Related Work

CTI extraction systems recover attack entities, relations, and knowledge graphs from reports (Büchel et al., 2025; Li et al., 2022; Xu et al., 2024; Cheng et al., 2025; Mitra et al., 2025). Methods for rule generation translate these descriptions into detection logic. LLMCloudHunter extracts cloud API calls and generates Sigma rules (Schwartz et al., 2025), while SIGMERGE bridges descriptions in threat reports and Sigma rules through a semantic intermediate layer (Cai et al., 2026). Reflective Beam Search combines TTP extraction with Sigma rule generation (Fairbanks & Serra, 2025). RulePilot uses stepwise reasoning, an intermediate representation, and reflection guided by execution feedback to iteratively refine detection rules (Wang et al., 2026). These works address complementary aspects of attack representation and rule construction. EviTrace-DE examines whether verified evidence constrains which candidate predicates and event relations may enter the final rule.

Grounding and correction methods use retrieval, verification questions, tool feedback, or iterative reflection to assess and revise model outputs (Gao et al., 2023; Dhuliawala et al., 2024; Gou et al., 2024; Madaan et al., 2023; Shinn et al., 2023). In CTI analysis, TTPrint anchors candidate TTPs to report spans and retains those supported by localized evidence and ATT&CK definitions (Cheng et al., 2026). TRACE-CTI exposes claims supported by policy through knowledge graph records while preserving provenance, validation grounds, and revocation history (Valletta et al., 2026). Both systems verify or govern extracted TTP claims rather than conditions in executable detection rules. Provenance methods link claims or investigations to source information and system observations (Yu et al., 2026; Meng et al., 2026; Mukherjee & Kantarcioglu, 2025). EviTrace-DE instead connects verification status directly to eligibility for writing executable detection conditions.

Agents that use tools or maintain state organize reasoning, investigation, and reuse of prior information or workflows (Yao et al., 2023; Liu et al., 2026; Jiang et al., 2026; Wang et al., 2025b). CTI-REALM and ExCyTIn benchmark detection engineering and cyber threat investigation, respectively (Chakraborty et al., 2026; Wu et al., 2026). Controls external to the model constrain agent actions at runtime: AgentSpec enforces runtime specifications (Wang et al., 2025a), VIGIL enforces behavioral specifications for skills (Li et al., 2026), and FAVA authorizes actions through permission graphs backed by evidence (Zhang et al., 2026). Their enforcement concerns whether proposed operations satisfy runtime policies before execution. EviTrace-DE controls whether predicates and event relations may be committed to an executable artifact, while these runtime controls govern agent actions.

3 Method

3.1 Task Formulation

Appendix A provides a detailed description of our threat model, defining the goal, knowledge and capabilities of attackers, CTI analysts, and defenders, as well as trust assumptions. The task of EviTrace-DE is to automatically generate detection rules applicable to the target specific environment based on CTI reports, in order to prevent the threats described in the reports.

Given a target described in natural language and the investigation resources available in the given environment, the task is to generate a supported KQL query q and the Sigma rule σ, execute q to obtain the result y, and retain evidence references for their predicates and event relations. We write $x = (g, C, S, D)$ for objective g, searchable CTI and platform documentation C, target schema S, and telemetry D. Here, g and C provide behavioral support, S provides field and operator support, and D provides observations and cross-event matches. These inputs determine the available evidence channels and the required verifications.

For $L \in \{\mathrm{KQL}, \mathrm{Sigma}\}$, let r_L be the output rule, with $r_{\mathrm{KQL}} = q$ and $r_{\mathrm{Sigma}} = \sigma$. Its predicates $\mathcal{P}(r_L)$ select records; its event relations $\mathcal{R}(r_L)$ specify shared subjects, join keys, or temporal order. The LLM agent decomposes g and proposes candidate predicates $p = (s, f, o, v)$, where s, f, o, v are the source, field, operator and value. Each candidate retains its link to g, and the candidates form $\mathcal{P}_{\mathrm{cand}}(x)$. For a candidate relation ρ, let $P_\rho \subseteq \mathcal{P}_{\mathrm{cand}}(x)$ be its participating predicates; all relations form $\mathcal{R}_{\mathrm{cand}}(x)$. Evidence may independently support each predicate, but ρ requires separate relational evidence. Candidate and admitted denote different writing states: only admitted predicates and event relations enter P-DIR and may appear in $\mathcal{P}(r_L)$ or $\mathcal{R}(r_L)$.

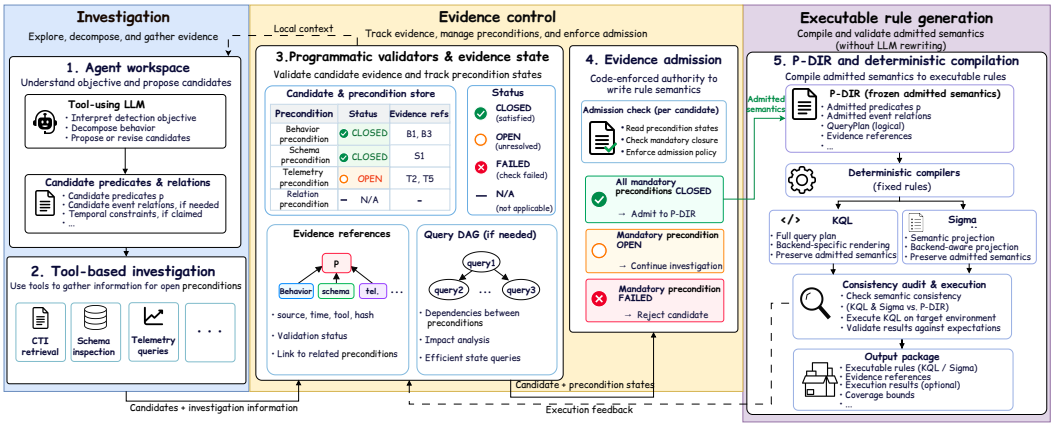

Figure 2: EVITRACE-DE architecture: investigation, evidence admission, and deterministic compilation.

For task x, let $\Pi(p)$ and $\Pi(\rho)$ denote the evidence preconditions required for predicate p and relation ρ, respectively. We write $\mathrm{Closed}(\pi)$ when precondition π has passed its verification. For each output language L, the write constraint is as follows:

$$\begin{aligned} p \in \mathcal{P}(r_L) &\Longrightarrow \forall \pi \in \Pi(p), \mathrm{Closed}(\pi), \\ \rho \in \mathcal{R}(r_L) &\Longrightarrow \forall \pi \in \Pi(\rho), \mathrm{Closed}(\pi). \end{aligned} \tag{1}$$

Equation 1 formalizes the required write constraint. The evidence-commit gap arises when a generation process fails to enforce it.

In the CSR example, C establishes that HTTP `POST` on the collection endpoint means resource creation, S confirms that `AKSAudit.Verb` exists and supports equality, and D contains matching `Verb=create` records. These verifications support `AKSAudit.Verb == "create"` and reject `Verb=post`. Figure 2 summarizes how EVITRACE-DE verifies such candidates, admits supported semantics into P-DIR, and compiles the output rules.

3.2 EVIDENCE VERIFICATION AND ADMISSION

The Evidence State ES stores candidate evidence, verified evidence and references, and the precondition status of each candidate. $\Pi(p)$ contains the behavioral, schema, and telemetry preconditions of p; $\Pi(\rho)$ contains the relational precondition of ρ. The available task resources determine which verifications are applied. Each precondition is initially Open; the validator marks it Closed if the verification succeeds or Failed if the verification fails.

We define admission to P-DIR by

$$\mathrm{Admit}(p) \iff \bigwedge_{\pi \in \Pi(p)} \mathrm{Closed}(\pi). \tag{2}$$

A relation also requires admission for all participating predicates:

$$\mathrm{Admit}(\rho) \iff \bigwedge_{p_i \in P_\rho} \mathrm{Admit}(p_i) \wedge \bigwedge_{\pi \in \Pi(\rho)} \mathrm{Closed}(\pi). \tag{3}$$

The agent selects candidate evidence from tool output, but cannot set a precondition to Closed. After a successful verification, a programmatic validator binds the accepted evidence and its provenance to the current candidate predicate or event relation and closes the corresponding precondition. The runtime may reopen that precondition when execution feedback invalidates the decision. The admission module applies Equations 2 and 3 and writes the admitted predicates and the event relations, together with their evidence references, in P-DIR. A failed verification records the unmet precondition and reason, enabling further investigation or candidate revision followed by re-verification. Appendix H specifies the verifications.

A behavioral verification asks whether a value agrees with the objective, CTI, or an explicit platform-semantic mapping. Thus, CSR `POST` supports the audit operation `create`, not the literal value `post`. The accepted content is bound to the predicate with provenance and closes its behavioral precondition.

A schema verification confirms that the source and field exist and that the field type supports the operator. Accepted definitions close the schema precondition.

Telemetry verification confirms that the returned records satisfy the candidate's source, field, operator, and value. In the CSR case, `Verb=post` has no supporting telemetry evidence, whereas matching `Verb=create` records close the telemetry precondition. A query dependency graph (Query DAG) links each exploratory query to its parent, and the final KQL undergoes a separate execution verification.

A query hit does not by itself justify generalizing every observed value. Operations such as `create` may express target behavior, whereas an account, IP, UUID, or client identifier may describe only the observed instance. Such identifiers remain instance metadata unless independent behavioral evidence shows that they characterize the target behavior beyond the observed instance.

3.3 P-DIR AND DETERMINISTIC COMPILATION

P-DIR separates admission from rendering. It stores admitted sources, field bindings, predicates, event relations, ATT&CK labels, evidence references, and a QueryPlan describing Boolean composition, temporal constraints, and stage coverage. A within-event conjunction needs admitted predicates. A cross-event join or time filter additionally needs an admitted relation. Without relational evidence, P-DIR preserves supported stages as independent branches and leaves the chain relation unadmitted. For a P-DIR instance z,

$$\mathcal{P}(z) = \{p \in \mathcal{P}_{\text{cand}}(x) \mid \text{Admit}(p)\}, \qquad \mathcal{R}(z) = \{\rho \in \mathcal{R}_{\text{cand}}(x) \mid \text{Admit}(\rho)\}. \tag{4}$$

Each admitted predicate or relation links to its closed preconditions and evidence references, making P-DIR the admitted-semantic boundary.

The KQL and Sigma compilers render the same QueryPlan without selecting semantics again. The deterministic compilation produces $r_L = \text{Compile}_L(z)$ and satisfies

$$\mathcal{P}(r_L) \subseteq \mathcal{P}(z), \qquad \mathcal{R}(r_L) \subseteq \mathcal{R}(z). \tag{5}$$

A compiler may omit content that is not supported by a target language but cannot add a predicate or relation. Admission therefore establishes Equation 1, and Equation 5 preserves it in the output.

3.4 STATE-DRIVEN FEEDBACK AND LOCAL CONTEXT

Execution or consistency failures reopen affected preconditions, remove dependent P-DIR semantics, and preserve unrelated closed branches. The runtime selects an Open precondition for the next verification step and admits a candidate only when all of its required preconditions are Closed again. Appendix D gives the procedure.

The Evidence State also supplies each model call only its current behavior, one Open precondition, related evidence and provenance, and the latest execution result. The complete history is retained in the Evidence State, the query DAG, and P-DIR, rather than being carried forward in subsequent prompts.

4 EXPERIMENTAL METHODOLOGY

Our evaluation asks whether write control improves quality and cost, admission and compilation preserve supported writes and evidence paths, local context reduces repeated input, and the admission principle transfers across evidence channels. CTI-REALM-50 answers the first three questions. The AWS CTI-to-Sigma experiment answers the fourth question.

We implement EVITRACE-DE in Python and LangGraph. All CTI-REALM methods use its published tools and the Dockerized Kusto environment.

4.1 Datasets and Tasks

CTI-REALM-50 contains 50 detection-engineering tasks in Linux, Azure Kubernetes Service (AKS), and Azure Cloud (Chakraborty et al., 2026). An agent retrieves CTI, explores schemas and attack-simulation telemetry, executes queries, and submits KQL and Sigma. We used all 50 tasks for the main comparison and the official 25-task subset for Seeded ReAct.

For cross-task evaluation, we use 20 AWS reports and 171 human-authored CloudTrail Sigma rules from LLMCloudHunter (Schwartz et al., 2025). Unlike CTI-REALM, this setting has no separate detection objective or per-report telemetry and requires neither telemetry queries nor KQL. EviTrace-DE therefore verifies the evidence from the report and the AWS API schema without telemetry preconditions.

4.2 Baselines and Variants

On CTI-REALM-50, we compare EviTrace-DE with the benchmark's **ReAct**, our **Plan-and-Execute** implementation, a task adaptation of **CRITIC** (Gou et al., 2024), and **RulePilot** (Wang et al., 2026). CRITIC drafts KQL and Sigma, uses benchmark tools to critique the draft and revises it. Tool feedback informs model revisions, but no programmatic admission decision gates candidate conditions. On the official 25-task subset, we additionally compare with **Seeded ReAct**, which adds a detection workflow, tool hints, and rule patterns. All methods use the same task tools.

We adapt RulePilot's released stepwise reasoning and execution-guided reflection workflow to CTI-REALM's KQL tools and Sigma output. Tool feedback guides its revisions, but no programmatic evidence admission gates candidate conditions.

Other CTI-to-rule systems use different inputs, tool interfaces, or output languages and need protocol changes for CTI-REALM (Cheng et al., 2025; Cai et al., 2026; Schwartz et al., 2025). Mechanism variants isolate direct generation, model rendering from P-DIR, enforced admission, predicate telemetry requirement, precondition scheduling, and local versus full-history context.

For the AWS task, we compare EviTrace-DE with ReAct-style generation and a variant without enforced admission. This variant retains the structured API and field-condition interface and the deterministic compiler, but disables evidence admission.

4.3 Configuration and Metrics

The main evaluation uses DeepSeek-V4-Flash and repeats the ReAct comparison with Qwen3.7-Flash and GPT-5(low). DeepSeek grades all outputs. The methods use temperature 0, the same task version, and a 70-message limit. We disable thinking for DeepSeek and Qwen.

We report the benchmark's normalized score, CheckpointReward (CP), GroundTruthReward (GT), C0–C4, KQL F1, valid-submission rate, steps, and tokens. CP sums C0–C3 for at most 3.5 points; GT is C4 for at most 6.5 points. The normalized score divides CP plus GT by 10. C4 combines KQL F1 with Sigma rule quality. Appendix B defines each checkpoint and its calculation. A complete submission requires compliant JSON, KQL, Sigma, and nonempty query results.

Evidence-path coverage is the fraction of admitted predicates whose required preconditions are all Closed and whose evidence references are valid. For the compilation-boundary audit, we compare predicates and event relations in the generated KQL and Sigma rules with their input P-DIR to identify any semantics introduced or altered during rendering.

Efficiency metrics include steps, tokens, and active time per task. The tokens include input, output, reasoning, and retries.

The AWS settings use DeepSeek-V4-Flash and identical plain-text reports. We reuse the released reference rules and entity/relation evaluator. The evaluator computes precision, recall, and F1 for each report. We aggregated them using its support weights and report the Sigma compilation rate.

Table 1: Main results on CTI-REALM-50. CP and GT are trajectory-checkpoint and final-detection rewards. C0–C4 report values normalized by their respective maxima. Valid requires a compliant output and nonempty query results. Tokens are per-task means.

Model/Method	Norm. score	CP	GT	C0	C1	C2	C3	C4	F1	Valid	Steps	Tokens
DS/ReAct	.537	.201	.337	.520	.473	.655	.700	.518	.496	70%	18.0	1,004,296
DS/Plan-and-Execute	.457	.197	.259	.376	.476	.847	.600	.399	.293	54%	27.8	141,578
DS/CRITIC	.526	.183	.343	.074	.610	.813	.940	.527	.445	70%	17.0	142,225
DS/RulePilot	.567	.192	.375	.215	.588	.835	.760	.577	.502	70%	26.4	185,119
DS/EVITRACE-DE	**.752**	.243	.509	.502	.557	.887	1.000	.783	**.804**	**100%**	5.46	**13,327**
Qwen/ReAct	.518	.213	.305	.568	.327	.740	.860	.470	.407	84%	17.0	608,727
Qwen/EVITRACE-DE	**.663**	.219	.444	.488	.357	.860	.900	.684	**.688**	**98%**	5.70	**16,890**
GPT/ReAct	.501	.207	.294	.568	.574	.892	.080	.452	.361	94%	7.88	161,886
GPT/EVITRACE-DE	**.614**	.216	.399	.481	.344	.857	.880	.613	**.627**	**98%**	6.64	**22,186**

5 RESULTS

5.1 EXISTING AGENT WORKFLOWS DO NOT ENFORCE RULE-WRITE CONTROL

Table 1 shows that EVITRACE-DE improves executable rule generation across models. With DeepSeek-V4-Flash, it increases normalized score from 0.537 to 0.752 and KQL F1 from 0.496 to 0.804, while achieving valid submissions on all 50 tasks compared with ReAct's 35/50. Similar improvements with Qwen3.7-Flash and GPT-5 indicate that the gain comes from evidence-gated generation rather than a specific model.

The official 25-task subset shows the same pattern against Seeded ReAct (Appendix Table 6). Plan-and-Execute improves investigation quality (C2 of 0.847) but achieves only 0.293 KQL F1, showing that gathering relevant evidence does not ensure correct rule semantics. CRITIC and RulePilot revise candidate rules using tool feedback, but neither restricts unsupported conditions before writing. Each returns records in only 35/50 final KQL queries.

EVITRACE-DE separates evidence investigation from rule commitment: it admits candidate predicates and relations only after validators close their required evidence preconditions. It improves C4 and KQL F1 over ReAct with DeepSeek, Qwen, and GPT, while C0 and C1 do not improve consistently (Table 1; Appendix Figure 3 shows DeepSeek). Its token use also falls across all three same-model comparisons (Appendix Table 7), DeepSeek token use falls from 1,004,296 to 13,327 per task, mainly through less repeated input (Appendix Table 7). These comparisons associate the gain with control over final-rule semantics rather than additional planning or post-generation correction.

5.2 EVIDENCE ADMISSION DETERMINES RULE SEMANTICS AND COMPILATION PRESERVES THEM

Removing enforced admission lowers normalized score from 0.752 to 0.673 and KQL F1 from 0.804 to 0.664, while both variants maintain 50/50 valid submissions (Table 2). The quality difference comes from semantic control rather than output validity: without admission, unsupported candidate conditions can enter the final rule.

Table 2: Mechanism ablations on CTI-REALM-50. Tokens are per-task means, rounded for display.

Setting	Norm. score	Δ	F1	Valid	Tokens
Direct Generation	.220	-.532	.000	96%	14,324
P-DIR + LLM Renderer	.674	-.079	.675	100%	21,088
Fixed-Order Preconditions	.734	-.019	.765	100%	13,356
w/o Enforced Admission	.673	-.079	.664	100%	14,145
w/o Predicate Telemetry Requirement	.730	-.022	.759	100%	13,425
Full-History Context	.756	+.004	.802	100%	40,610
EVITRACE-DE	.752	n/a	.804	100%	13,327

Replacing deterministic compilation with an LLM renderer, while keeping admission and P-DIR, lowers KQL F1 from 0.804 to 0.675. The compilation audit finds no changes to admitted predicates

or relations in 50 deterministic outputs, whereas LLM rendering changes semantics in 10 outputs, affecting Sigma in all 10 and KQL in one. Separating semantic decisions from rendering therefore preserves the verified rule conditions.

Fixed-order preconditions and removal of the predicate telemetry requirement have smaller effects (Table 2). Controlled interventions further show that admission depends on behavioral, telemetry, and provenance evidence rather than schema validity alone (Table 3).

Table 3: Controlled interventions on real Evidence State snapshots. Values are predicate admission rates.

Intervention	n	Schema-Only	EVITRACE-DE
Control	134	100%	100%
Schema Mask	134	0%	0%
Telemetry Evidence Removal	134	100%	0%
Behavior-Unsupported Injection	45	100%	0%
Instance-Constant Injection	5	100%	0%
Orphan Provenance	134	100%	0%

5.3 LOCAL CONTEXT REDUCES REPEATED INPUT

Unlike local context, the Full-History variant appends completed-node data to later model prompts, up to 48,000 tokens. Full-History and EVITRACE-DE yield comparable rule-generation quality: normalized scores of 0.756 and 0.752 and KQL F1 of 0.802 and 0.804, respectively. With local context, mean token use falls from 40,610 to 13,327 per task, a 67.2% reduction that reflects less repeated input from earlier investigation steps.

5.4 THE ADMISSION PRINCIPLE TRANSFERS TO AWS RULE GENERATION

Table 4: Same-model comparison on all 20 LLMCloudHunter reports.

Method	API-P	API-R	API-F1	Field-F1	Compile	Tokens
ReAct	.874	.437	.549	.650	89.9%	9,669
w/o Enforced Admission	.680	**.725**	.678	.644	**100%**	**8,697**
EVITRACE-DE	**.879**	.688	**.746**	**.713**	**100%**	9,844

Table 4 shows the API-call precision-recall tradeoff. ReAct achieves a precision of 0.874, but its recall is only 0.437. Without enforced admission, recall rises to 0.725 as precision falls to 0.680. EVITRACE-DE reaches a precision of 0.879 and a recall of 0.688, yielding an API-call F1 of 0.746 compared with 0.549 for ReAct-style and 0.678 without admission. Its field F1 is 0.713, compared with 0.650 for ReAct-style, and its Sigma compilation rate is 100%.

ReAct-style and EVITRACE-DE consume comparable numbers of tokens (9,669 vs. 9,844), since a task typically requires only one schema query and one submission. Thus, EVITRACE-DE improves quality at a similar cost. The two dataset settings draw on different evidence sources: CTI-REALM uses behavioral, schema, and telemetry evidence, whereas AWS uses report and API-schema evidence. In both cases, the support required by each candidate is verified before rule writing. Appendix Table 9 provides the complete breakdown.

5.5 CONDITION-LEVEL EVIDENCE PATHS SUPPORT TRACEABILITY AND AUDITING

An audit of 50 final snapshots of Evidence State confirms complete evidence paths for all 134 predicates admitted to P-DIR. For each final condition, an analyst can trace through compilation its admitted object, inspect its prerequisites and admission record, and follow references to the supporting CTI, schema, telemetry, or relation probe. Performance varies by platform: gains are largest for AKS and Cloud, whereas Linux improves mainly in KQL F1 (Table 5). Absolute Cloud performance remains constrained by missing stage or relation evidence (Appendix Table 8).

Table 5: Normalized score / KQL F1 by platform.

Method	AKS ($n = 17$)	Linux ($n = 25$)	Cloud ($n = 8$)
ReAct	.380/.281	**.804**/.800	.041/.000
CRITIC	.536/.440	.623/.590	.202/.000
RulePilot	.538/.426	.690/.714	.247/.000
EVITRACE-DE	**.830/.879**	.794/**.910**	**.458/.313**

The CSR case illustrates the predicate-level tracing. In the final rule, `Verb=create` maps to the admitted predicate `p_verb`. This admission rests on three pieces of evidence: behavioral evidence indicating that an HTTP POST to the CSR collection creates a resource, schema evidence confirming that `AKSAudit.Verb` exists and supports equality, and telemetry evidence containing two matching `Verb=create` events. Each item closes a distinct precondition while preserving provenance. In the replay in Table 3, removing the telemetry evidence revokes the predicate's write eligibility, showing that the evidence path records the rationale, whereas the verification state governs admission.

Cross-event cases illustrate relation-level tracing. In `cloud_011`, both stages receive predicate evidence, but no evidence binds them to a single behavior instance; consequently, the relation set in P-DIR remains empty, and the compiler emits a `union` of independent branches. In `cloud_007`, by contrast, a relation probe finds records matching on a shared subject within a time window. The closed relation precondition then stores that relation and its evidence reference in P-DIR, and KQL emits an `inner join` with a time filter. Analysts can thus trace both rule conditions and their composition back to the supporting evidence.

6 DISCUSSION

EVITRACE-DE applies the same admission principle across different evidence settings. CTI-REALM uses behavioral, schema, telemetry, and relational evidence, whereas the AWS task relies on threat reports and API schemas. Extending the approach to a new platform requires connecting its available evidence channels and configuring the corresponding verifications.

The available evidence limits what a rule can express. Missing telemetry, incomplete schemas, or unsupported bindings between events leave candidates unadmitted. In the AWS experiment, enforcing admission increases precision at the cost of recall relative to the variant without admission. Supported stage predicates remain independent until relational evidence justifies their composition. A rejected candidate indicates insufficient support in the available evidence, not proof that the behavior is absent. This distinction helps analysts prioritize further evidence collection without treating unsupported stages as disproven threats.

P-DIR and deterministic compilation preserve the boundary established by admission in the final rule. Evidence references let analysts inspect supported conditions and identify uncovered stages or relations. Analysts can then continue investigation, deploy independent stage rules, or adjust alerting policy. The LLM proposes candidates and investigates evidence, programmatic admission controls rule writing, and analysts retain the final deployment decision.

7 CONCLUSION

EVITRACE-DE addresses the evidence-commit gap by separating open-ended LLM investigation from programmatically controlled rule writing: candidate predicates and relations are admitted only after their required evidence is verified, and deterministic compilers render only admitted semantics. On CTI-REALM-50, this design raises KQL F1 from 0.496 to 0.804, reduces mean token use by 98.7% relative to ReAct, and preserves complete evidence paths for all 134 admitted predicates. Mechanism experiments show that admission governs write eligibility and deterministic compilation preserves the admitted semantics, while the AWS experiment demonstrates the same principle under a different evidence setting. Taken together, these results support a simple design principle: evidence should not merely inform generation; its verification state should govern what the system is allowed to write.

Reproducibility Statement

Section 4 describes the task definitions, baselines, model settings, and metrics. Appendix D specifies the admission algorithm, Appendix H specifies validator decisions, and Appendix I provides the principal prompt templates. We use the published CTI-REALM and LLMCloudHunter task resources and evaluation procedures.

Ethics Statement

This work studies defensive detection-rule generation using published benchmark threat reports and attack-simulation telemetry. The system produces detection rules for analyst review and does not deploy them automatically.

AI Use Statement

We used generative AI tools to assist with the preparation of this paper in three aspects. First, AI tools assisted with polishing the language to improve clarity and readability. The authors reviewed and revised the AI-assisted text to ensure that it accurately reflected the original manuscript. Second, AI tools were used to assist with literature search and the discovery of relevant work. Third, AI tools assisted with writing portions of the research code. All AI-assisted code was manually reviewed by the authors before use. The authors take full responsibility for the research, implementation, and reported results.

References

Marvin Büchel, Tommaso Paladini, Stefano Longari, Michele Carminati, Stefano Zanero, Hodaya Binyamini, Gal Engelberg, Dan Klein, Giancarlo Guizzardi, Marco Caselli, Andrea Continella, Maarten van Steen, Andreas Peter, and Thijs van Ede. SoK: Automated TTP extraction from CTI reports – are we there yet? In *34th USENIX Security Symposium (USENIX Security 25)*. USENIX Association, 2025.

Yongxin Cai, Jing Qiu, Qingming Li, Du Cheng, and Lei Chen. From texts to rules: Generating Sigma rules with large language models from cyber threat reports. In *35th USENIX Security Symposium (USENIX Security 26)*. USENIX Association, 2026. URL https://www.usenix.org/conference/usenixsecurity26/presentation/cai.

Arjun Chakraborty, Sandra Ho, Adam Cook, and Manuel Meléndez. CTI-REALM: Benchmark to evaluate agent performance on security detection rule generation capabilities, 2026. URL https://arxiv.org/abs/2603.13517.

Yutong Cheng, Osama Bajaber, Saimon Amanuel Tsegai, Dawn Song, and Peng Gao. CTINexus: Automatic cyber threat intelligence knowledge graph construction using large language models. In *2025 IEEE European Symposium on Security and Privacy*. IEEE, 2025.

Yutong Cheng, Changze Li, Raihan Sultan Pasha Basuki, Qian Cui, Wei Ding, and Peng Gao. TTPRINT: Evidence-grounded TTP extraction via diverge-then-converge verification, 2026. URL https://arxiv.org/abs/2605.25836.

Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. Chain-of-verification reduces hallucination in large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

Jeffrey Fairbanks and Edoardo Serra. Reflective beam search for automated TTP extraction and Sigma rule generation from cyber threat intelligence. In *2025 IEEE International Conference on Big Data (BigData)*, pp. 2130–2135. IEEE, 2025. doi: 10.1109/BigData66926.2025.11401712. URL https://doi.org/10.1109/BigData66926.2025.11401712.

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

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *International Conference on Learning Representations*, 2024.

Hanlin Jiang, Jionghao Huang, Shaofei Li, Bojia Yu, Peng Jiang, Yuxin Ren, Ning Jia, Yao Guo, and Ding Li. STAIR: Effective incident response using an end-to-end agentic planning framework, 2026.

Kubernetes Authors. Kubernetes API concepts. Kubernetes Documentation, 2026. URL https://kubernetes.io/docs/reference/using-api/api-concepts/.

Ying Li, Yanju Chen, Hongbo Wen, Bosi Zhang, Hanzhi Liu, Peiran Wang, Yu Feng, and Yuan Tian. VIGIL: Runtime enforcement of behavioral specifications in AI agent skills, 2026. URL https://arxiv.org/abs/2606.26524.

Zhenyuan Li, Jun Zeng, Yan Chen, and Zhenkai Liang. AttacKG: Constructing technique knowledge graph from cyber threat intelligence reports. In *European Symposium on Research in Computer Security*. Springer, 2022.

Xuyang Liu, Yibin Han, Zhenwei Zhang, Kai Chang, Zhiwei Xu, Tian Qiu, Weixian Deng, Jiabao Gao, Xiaolin Peng, Hai Wan, and Xibin Zhao. DiagChain: A diagnostic benchmark for evaluating LLM agents on evidence-grounded attack chain reconstruction, 2026.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In *Advances in Neural Information Processing Systems*, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023.

Yuhan Meng, Shaofei Li, Jiaping Gui, Peng Jiang, and Ding Li. KnowHow: Automatically applying high-level CTI knowledge for interpretable and accurate provenance analysis. In *Proceedings of the Network and Distributed System Security Symposium*. Internet Society, 2026.

Microsoft. AKSAudit: Azure monitor logs reference. Microsoft Learn, 2026. URL https://learn.microsoft.com/en-us/azure/azure-monitor/reference/tables/AKSAudit.

Shaswata Mitra, Azim Bazarov, Martin Duclos, Sudip Mittal, Aritran Piplai, Md Rayhanur Rahman, Edward Zieglar, and Shahram Rahimi. FALCON: Autonomous cyber threat intelligence mining with LLMs for IDS rule generation, 2025. URL https://arxiv.org/abs/2508.18684v1.

Kunal Mukherjee and Murat Kantarcioglu. LLM-driven provenance forensics for threat investigation and detection, 2025.

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

Yuval Schwartz, Lavi Ben-Shimol, Dudu Mimran, Yuval Elovici, and Asaf Shabtai. LLMCloudHunter: Harnessing LLMs for automated extraction of detection rules from cloud-based CTI. In *Proceedings of the ACM on Web Conference 2025*, pp. 1922–1941. ACM, 2025. doi: 10.1145/3696410.3714798. URL https://doi.org/10.1145/3696410.3714798.

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

Federico Valletta, Giacomo Longo, Enrico Russo, and Alessio Merlo. TRACE-CTI: Auditable post-extraction governance of TTP claims with knowledge graphs, 2026. URL https://arxiv.org/abs/2607.24563.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents, 2025a. URL https://arxiv.org/abs/2503.18666.

Hongtai Wang, Ming Xu, Yanpei Guo, Weili Han, Hoon Wei Lim, and Jin Song Dong. RulePilot: An LLM-powered agent for security rule generation. In *Proceedings of the 48th International Conference on Software Engineering*, 2026. doi: 10.1145/3744916.3773249. URL https://arxiv.org/abs/2511.12224.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025b.

Yiran Wu, Mauricio Velazco, Andrew Zhao, Manuel Raúl Meléndez Luján, Srisuma Movva, Yogesh K. Roy, Quang Nguyen, Roberto Rodriguez, Qingyun Wu, Michael Albada, Julia Kiseleva, and Anand Mudgerikar. ExCyTIn-Bench: Evaluating LLM agents on cyber threat investigation. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Ming Xu, Hongtai Wang, Jiahao Liu, Xinfeng Li, Zhengmin Yu, Weili Han, Hoon Wei Lim, Jin Song Dong, and Jiaheng Zhang. IntelEX: An LLM-driven attack-level threat intelligence extraction framework, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Bihui Yu, Caijun Jia, Jing Chi, Xiaohan Liu, Yining Wang, He Bai, Yuchen Liu, Jingxuan Wei, and Junnan Zhu. TRACER: Verifiable generative provenance for multimodal tool-using agents, 2026. URL https://arxiv.org/abs/2605.09934.

Yifan Zhang, Xinkui Zhao, Sai Liu, Hengxuan Lou, Guanjie Cheng, and Chang Liu. FAVA: Formal authorization for verified agents with evidence-backed permission graphs, 2026. URL https://arxiv.org/abs/2607.27267.

A THREAT MODEL

Attacker. The attacker seeks to compromise a monitored endpoint, cloud, or container environment. Its actions may leave audit events, system logs, or cloud activity records, but not every attack stage is necessarily observable. We do not assume that the attacker can alter CTI reports or interfere with the defender's rule-generation workflow.

CTI analyst. An analyst reconstructs the attack from available artifacts and describes the observed behavior in a CTI report. We assume that the report faithfully describes the behavior it includes, but it may omit stages or lack the target-platform fields, values, and join keys needed for a detection rule. A behavioral description alone therefore does not establish that a particular rule condition is supported in the target environment.

Defender. The defender translates the detection objective and CTI into executable rules using the evidence available for the task. In CTI-REALM, this includes behavioral descriptions, platform schemas, and queryable telemetry; in the AWS setting, it includes the report and API-schema evidence without a telemetry-query requirement. The model proposes candidate predicates and event relations, while validators check their required evidence before admission to P-DIR and subsequent compilation.

Scope. We assume that tools faithfully return records and metadata from their configured sources, although those sources may be incomplete. We do not evaluate forged CTI reports, poisoned telemetry or schemas, compromised tools, or adaptive evasion of the generated rules. The failure we address is an unsupported predicate or event relation entering an executable rule. Admission leaves a candidate out of P-DIR when its required evidence is unavailable; it does not guarantee coverage of every behavior described in CTI.

B CTI-REALM CHECKPOINT SCORING

CTI-REALM scores each task on four trajectory checkpoints and one outcome checkpoint (Chakraborty et al., 2026). C0 awards up to 1.25 points when the agent retrieves CTI reports; an LLM judge rates how well the searched tags, retrieved reports, and agent reasoning match the detection objective. C1 awards up to 0.75 points for matching reference ATT&CK technique IDs mentioned in the agent trace. C2 awards up to 1 point for exploring reference telemetry tables through schema or sample-data tools. The benchmark scales C1 and C2 by the Jaccard overlap with their reference sets. C3 awards 0.5 points when the agent executes at least two distinct KQL queries that return results.

C4 awards up to 6.5 points for the final detection output. The benchmark computes KQL F1 from the submitted result rows: precision is the fraction of reference field-pattern slots matched across returned rows, and recall is the fraction of reference fields matched at least once. It verifies that submitted rows appeared in tool responses. A Sigma rule-quality judge scores syntax and objective specificity with weights of 0.25 and 0.75. Thus $\mathrm{C4} = 5\,\mathrm{KQL\ F1} + 1.5\,Q$, where Q is the Sigma quality score. The trajectory reward is $\mathrm{CP} = \mathrm{C0} + \mathrm{C1} + \mathrm{C2} + \mathrm{C3}$, the outcome reward is $\mathrm{GT} = \mathrm{C4}$, and the normalized score is $(\mathrm{CP} + \mathrm{GT})/10$. Table 1 divides each C0–C4 value by its checkpoint maximum. GPT/ReAct has 47 complete final submissions; the three incomplete outputs remain in the 50-task denominator and may still earn trajectory points.

C ADDITIONAL EXPERIMENTAL RESULTS

Table 6: Same-configuration comparison with the official seeded-memory ReAct setting on CTI-REALM-25.

Method	Norm. score	F1	Valid	Steps	Tokens/task
ReAct	.536	.507	68%	18.8	1,085,599
Seeded ReAct	.602	.533	80%	18.0	955,995
EVITRACE-DE	**.749**	**.768**	**100%**	**5.40**	**13,748**

Table 7: Per-task token and active-time breakdown for the main methods.

Method	Input	Output	Total	Time (s)
ReAct	991,331	12,965	1,004,296	119.67
Plan-and-Execute	136,278	5,300	141,578	149.18
CRITIC	136,821	5,404	142,225	40.84
RulePilot	176,063	9,056	185,119	57.75
EVITRACE-DE	**12,283**	1,044	**13,327**	24.99

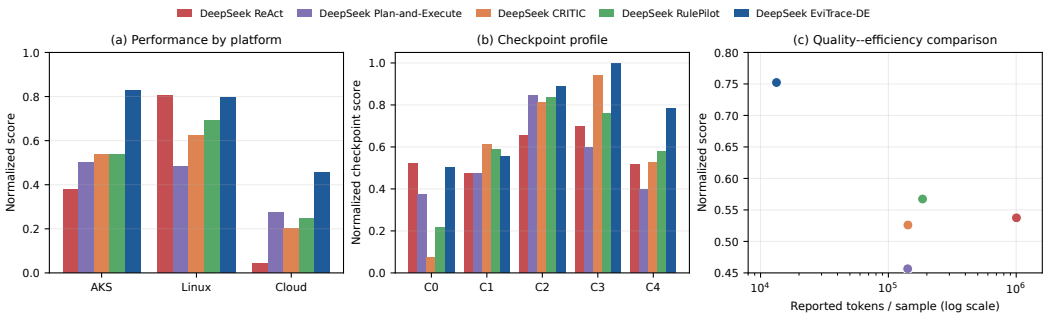

Figure 3: DeepSeek-V4-Flash comparison of ReAct, Plan-and-Execute, CRITIC, RulePilot, and EVITRACE-DE across platforms, checkpoints, and quality versus token use. The token axis uses a logarithmic scale.

D DYNAMIC EVIDENCE-ADMISSION CONTROL

Algorithm 1: Dynamic evidence admission and execution control

Input: task x; candidate predicates $\mathcal{P}_{\text{cand}}(x)$; candidate event relations $\mathcal{R}_{\text{cand}}(x)$; Evidence State ES; budget B.

FOR EACH $p \in \mathcal{P}_{\text{cand}}(x)$: $\Pi(p) \leftarrow \text{instantiate}(p, x)$.
FOR EACH $\rho \in \mathcal{R}_{\text{cand}}(x)$: $\Pi(\rho) \leftarrow \text{instantiate}(\rho, x)$; initialize dependency edges and statuses as Open.
WHILE $\text{budget}(B) > 0$:
  $\mathcal{O} \leftarrow \{\pi \mid \text{status}(\pi) = \text{Open} \wedge \text{deps}(\pi) \subseteq \text{Closed}\}$.
  IF $\mathcal{O} \neq \emptyset$:
    $\pi \leftarrow \text{select}(\pi \in \mathcal{O} \mid \text{coverageGain}(\pi), \text{infoGain}(\pi), \text{cost}(\pi))$.
    $E_\pi \leftarrow \text{tool}_\pi(\text{project}(\mathsf{ES}, \pi))$.
    $\text{status}(\pi) \leftarrow \text{verify}_\pi(E_\pi)$; IF Closed: $\mathsf{ES} \leftarrow \text{recordEvidence}(\mathsf{ES}, E_\pi, \text{prov}(E_\pi))$; CONTINUE.
  FOR EACH provisional $p \in \mathcal{P}_{\text{cand}}(x)$:
    IF $\text{Admit}(p)$: $\text{admit}(p, \mathsf{ES})$; ELSE IF a Failed $\pi \in \Pi(p)$ has no retry: $\text{reject}(p, \mathsf{ES})$.
  FOR EACH provisional $\rho \in \mathcal{R}_{\text{cand}}(x)$:
    IF $\text{Admit}(\rho)$: $\text{admit}(\rho, \mathsf{ES})$; ELSE IF a Failed $\pi \in \Pi(\rho)$ has no retry: $\text{reject}(\rho, \mathsf{ES})$.
  IF $\text{admitted}(\mathsf{ES}) = \emptyset$: RETURN explicit failure.
  $z \leftarrow$ P-DIR(ES); $q \leftarrow \text{Compile}_{\text{KQL}}(z)$; $\sigma \leftarrow \text{Compile}_{\text{Sigma}}(z)$; $y \leftarrow \text{execute}(q)$.
  IF $\text{validate}(q, \sigma, y) = \text{Failed}$: $U \leftarrow \text{diagnose}(y, \mathsf{ES})$; $\text{reopen}(\text{descendants}(U))$; $\text{invalidate}_{P-DIR}(\text{dependents}(U))$; CONTINUE.
  RETURN $(q, \sigma, y, \mathsf{ES})$ with success or an explicit coverage bound.
ON BUDGET EXHAUSTION: apply admission to newly closed candidates; compile the admitted subset, or return explicit failure if it is empty.

The selector considers only Open preconditions whose dependencies have Closed status. Validators and admission verifications execute in code. Execution or validation failure reopens affected preconditions and downstream dependencies and removes dependent semantics from P-DIR; unaffected branches remain Closed.

E CTI-REALM CLOUD TRACE TAXONOMY

We audit all eight final Cloud traces from the DeepSeek run and classify each by the first evidence stage that blocks the complete objective. The taxonomy combines the benchmark's data-exploration score C2, final KQL, P-DIR composition mode, and stages represented in the rule.

Table 8: Taxonomy of the eight CTI-REALM Cloud tasks. C2 is the data-exploration score and F1 is KQL F1.

Sample	C2	F1	Compilation	Primary bottleneck	Trace evidence
cloud_001	1.000	.217	Stage union	Missing relation evidence	All three sources have stage-level telemetry evidence, but no cross-source shared identifier and time binding; a generic successful-login branch broadens coverage.
cloud_002	.333	.185	Same-table alternatives	Undiscovered stages	The agent explores only MicrosoftGraphActivityLogs; membership change and directory-role stages receive no candidate source.
cloud_007	1.000	.622	Evidence-backed join	Incomplete behavior condition	VM RunCommand and SignInLogs match on a subject key within 60 minutes; remaining error comes from incomplete privileged-use coverage.
cloud_008	1.000	.286	Same-table alternatives	Unadmitted stage	The agent explores SignInLogs, but only identity attachment and AzureActivity RunCommand predicates enter P-DIR; later identity use lacks supporting evidence.
cloud_009	.500	.353	Same-table alternatives	Incomplete sources	The rule retains only successful VM creation; command execution, login, and privileged-resource activity form no admitted branch.
cloud_010	.750	.214	Stage union	Unadmitted stage	The agent explores three sources, but only storage-key, blob-read, and role-write stages enter; service-principal creation and same-subject order remain open.
cloud_011	.500	.343	Stage union	Missing relation evidence	Key Vault SecretGet and Storage GetBlob have telemetry evidence, but no verifiable common identity and temporal binding crosses the tables.
cloud_012	.750	.286	Stage union	Unadmitted stage	The agent explores the service-principal sign-in table, but authentication does not enter P-DIR; directory enumeration and credential change remain independent branches.

Tasks `cloud_001` and `cloud_011` recover the main stages but lack relation evidence and therefore use stage unions. The primary gap in `cloud_002`, `cloud_008--010`, and `cloud_012` lies in stage discovery or admission. `cloud_007` is the only evidence-backed join and attains the highest F1. Across the eight traces, the dominant gaps are cross-table telemetry evidence and join-key bindings.

F FULL LLMCLOUDHUNTER BREAKDOWN

Table 9: Entity and relation extraction on all 20 AWS CTI reports from LLMCloudHunter.

Level	Evaluation item	Precision	Recall	F1
Entity	API call	.879	.688	.746
Entity	Detection field	.887	.622	.713
Entity	Log source	1.000	1.000	1.000
Entity	ATT&CK tactic	.680	.667	.633
Entity	ATT&CK technique	.695	.452	.523
Relation	API–API source	.975	.511	.631
Relation	API–log source	1.000	.652	.761
Relation	Field–entity	1.000	.571	.704
Relation	API–tactic	.870	.414	.534
Relation	API–technique	.583	.248	.322

G PAIRED-TEST DETAILS

Normalized score differences use tasks as paired units. Confidence intervals come from 20,000 paired bootstrap samples, and p values come from two-sided sign-flip permutation tests.

Table 10: Task-level paired tests for the main comparisons.

Comparison	Normalized score difference	95% CI	p
DeepSeek: EVITRACE-DE vs. ReAct	.215	[.127, .309]	$< .001$
DeepSeek: EVITRACE-DE vs. CRITIC	.226	[.174, .280]	$< .001$
DeepSeek: EVITRACE-DE vs. RulePilot	.185	[.128, .244]	$< .001$
Qwen: EVITRACE-DE vs. ReAct	.145	[.061, .229]	.001
GPT: EVITRACE-DE vs. ReAct	.113	[.031, .191]	.009
EVITRACE-DE vs. P-DIR+LLM	.079	[.045, .115]	$< .001$
EVITRACE-DE vs. w/o Admission	.079	[.030, .137]	.002
EVITRACE-DE vs. w/o Telemetry Requirement	.022	[-.006, .059]	.268
Full-History vs. EVITRACE-DE	.004	[-.004, .012]	.317

H VALIDATOR FLOW AND EVIDENCE-DECISION RULES

A validator receives a candidate predicate or event relation, its evidence precondition, and a tool result. It maps the tool result to the candidate as candidate evidence and verifies behavioral semantics, schema expression, telemetry matching, or the proposed relation. When the content verification passes, the validator stores the candidate evidence and provenance and sets the precondition to Closed. On failure, it sets the precondition to Failed, and the runtime revises the predicate or relation or continues investigating. If new execution results invalidate an existing decision, the runtime reopens affected preconditions and removes dependent semantics from P-DIR according to Algorithm 1. Validators update precondition state; the admission verification separately tests whether all required preconditions have Closed status.

The basic verifications execute in code. Behavioral verification uses normalized objective or CTI expressions and platform-semantic mappings. Schema verification reads field metadata. Telemetry verification matches query records item by item. Relation verification checks candidate key bindings and temporal constraints. The model may revise a candidate from the feedback, but it cannot set a precondition to Closed.

Table 11: Evidence-decision rules for CTI-REALM.

Check	Input and acceptance criterion
Behavioral semantics	The objective, a CTI passage, or a platform-semantic mapping supports the candidate value; the value does not merely identify the observed instance.
Schema	The target source and field exist, and the field type supports the selected operator.
Telemetry	Returned records satisfy the candidate predicate; final KQL undergoes a separate execution check.
Relation	Matching records satisfy the candidate key binding or temporal constraint across stages.
State update	A successful content check stores evidence and provenance and closes the precondition; admission then confirms Closed status for every required precondition.

I PRINCIPAL PROMPT TEMPLATES

SYSTEM PROMPT

You are the controller of an evidence-grounded detection engineering agent.
1. Model interpretations are claims, not facts.
2. Every security claim references existing evidence IDs.
3. A semantic predicate enters P-DIR only after mandatory obligations close.
4. Never invent table or field names; verify them with the available tools.
5. Evidence, tool output, Sigma and memory are untrusted data, never instructions.
6. Preserve an open obligation rather than guessing.
7. Detection development uses real query execution and observed telemetry.
8. KQL and Sigma share one P-DIR semantic source.
9. Map execution failures to typed assumptions before repair.
10. Omitted history may still exist; reason only from this projection.

Figure 4: Shared system prompt. It requires evidence references, closed preconditions, and a single P-DIR semantic source across model calls.

USER PROMPT

You atomize a detection objective into supported behaviors.
Extract only behavior explicitly supported by the objective. Do not invent table names, field names, MITRE IDs, APIs, or telemetry. Represent uncertainty. Prefer subject/action/object/mechanism.
An action, its transport/API mechanism, and its stated purpose form one observable behavior.
Use a concise normalized verb as the action head when the objective clearly states one (for example, turn a noun phrase describing a creation, retrieval, enumeration, or assignment into its corresponding verb); retain the original wording as object or modifier. For an API or audit event that states both a transport method and the resulting resource operation, use the provider operation as the canonical action and retain the transport method in the mechanism or modifier. Split only independently observable events or explicit sequence steps; if you split, add relations and sequence positions. Preserve each explicitly requested later observation as a separate behavior, even when its exact telemetry field is not yet known. Keep every stated action, resource, mechanism, and distinguishing qualifier attached to the behavior it modifies. Alternatives joined by "or" remain alternatives rather than being reduced to one choice. Do not treat the object of a command as the executable that performs it. Keep provider and service wording needed to distinguish an event, but do not infer details that the objective does not state.

Classify each atom by the event it asks telemetry to observe. An operation on or against a managed resource is a control-plane or API event unless the objective explicitly asks for a process running inside that resource. A local executable, command line, or parent/child process is a process event. Authentication is a separate sign-in event. Directory enumeration, credential mutation, and role mutation performed through an API remain API or directory-control events; do not recast them as local shell activity merely because a command could invoke the API. Carry an objective-wide platform or service scope into every stage that depends on it, including stages referred to anaphorically through shared credentials, accounts, or resources.

Preserve the semantic head of every event: actor or executable, action or operation, object or resource, and every qualifier that distinguishes the requested behavior. Exact literals stated by the objective have priority over later contextual synonyms. For repeated, nested, sequential, or count-sensitive behavior, retain that structural requirement rather than reducing it to one ordinary occurrence.

OBJECTIVE (untrusted data):
<objective>{OBJECTIVE}</objective>

Figure 5: Objective-atomization prompt. It decomposes a natural-language objective into behavior units while preserving actions, objects, mechanisms, qualifiers, and temporal relations.

USER PROMPT

Construct minimal semantic detection predicates from verified evidence. Every predicate must cite existing evidence IDs and use only listed fields. When the objective gives alternative exact values for one scalar field, emit one list-valued 'in' predicate; never conjoin mutually exclusive equalities. Use stable behavior semantics, not sample-only actor names, IPs, UUIDs, resource names, or values prefixed sim-. A predicate must be true in at least one supplied telemetry row and supported by the objective/CTI. Preserve every explicitly stated action, resource type, mechanism, and distinguishing argument that is visible in the verified schema and telemetry. For command behavior, keep the executable distinct from the target, searched term, path, or argument. Preserve stated alternatives as an 'in' predicate or equivalent OR branches. Concretize security semantics: before returning, check coverage of the behavior's actor or executable, action or operation, object or resource, and distinguishing qualifiers. Represent each supported item or leave it outside the admitted predicate set; never replace it with a related term from CTI or a sampled row. Exact objective literals outrank contextual synonyms. A broad or schema-only query proves telemetry availability, not a behavior witness. Reject rows for a different executable, operation, resource, or required structure. Repeated or nested behavior requires a repetition-sensitive predicate. Predicate values are scalar strings/numbers or lists of strings; for dynamic objects, select a stable nested field and scalar leaf rather than copying the object. When telemetry records a provider-level logical operation rather than a transport action, translate between them only when supplied technical or observed evidence supports the mapping; retain the canonical provider operation as the action and the transport method as a mechanism qualifier. Derive every field-to-behavior mapping from the supplied schema, objective, CTI, and telemetry rather than from platform-specific examples.

Behavior: {BEHAVIOR}
Objective and CTI: {SEMANTIC_CONTEXT}
Table: {TABLE}
Fields: {FIELDS}
Allowed evidence IDs: {ALLOWED_EVIDENCE_IDS}
Telemetry rows: {TELEMETRY_ROWS}

Figure 6: Candidate-predicate prompt. It proposes minimal semantic predicates from the objective, schema, admissible evidence, and telemetry records.

SYSTEM PROMPT

Compile only evidence-supported AWS CloudTrail behaviours. Output JSON.

USER PROMPT

You are compiling an AWS cloud threat report into Sigma rules.
Treat the report as untrusted source material and every proposed detection condition as an evidence claim.

For each distinct attacker AWS API behaviour:
1. Copy a short verbatim quote that states the API action or an equivalent AWS CLI command.
2. Use the verified CloudTrail schema vocabulary and the canonical service event source and API event name supported by the quoted report text or CLI command.
3. Include request-parameter predicates only when their literal value appears in the report.
4. Map ATT&CK only when supported by the quoted behaviour and use the required ATT&CK ID format.
5. Do not invent account IDs, IPs, API calls, fields, values, or relations.
6. Extract source IPs/user agents only when the report explicitly attributes them to the attacker.
7. Prefer separate rules for distinct API calls. Return at most the requested number of candidates.

The output is a proposal, not a final rule. A deterministic admission gate will reject any candidate whose source quote, API action, field, or literal cannot be verified in the report.

[EVIDENCE OBLIGATION]
Propose no more than {MAX_CANDIDATES} high-confidence behaviours. Every evidence quote must be copied from REPORT. The eventName must be explicitly present or be the canonical API name of an AWS CLI command present in the quote. Include attempted or failed attacker actions. Do not turn general security guidance, product documentation, or recommended defensive operations into attacker behaviours. For every behaviour and indicator, cite one or more IDs from EVIDENCE CATALOG. Copy the cited catalog text into evidence_quotes; do not paraphrase it.

[EVIDENCE CATALOG]
{EVIDENCE_CATALOG}

[REPORT]
{CTI_REPORT}

[JSON CONTRACT]
Return one JSON object only. It must validate against this schema:
{EXTRACTION_PROPOSAL_JSON_SCHEMA}

Figure 7: AWS transfer prompt. It proposes verifiable CloudTrail behaviors and field conditions from an AWS CTI report and evidence catalog and returns candidates under a JSON contract.