GenAI Cybersecurity Copilot (PoC) – Final Submission

Scope: RAG-based SOC copilot that summarizes incident reports, retrieves similar incidents, and suggests mitigation steps.

Prepared: January 22, 2026

What this document answers (How / Why / How Much)

- **How** it works end-to-end (flow + methods used).
- **Why** we made each architectural choice (alternatives + trade-offs).
- **How much** it costs (token/cost method + telemetry + conceptual breakdown).

Submission checklist (mapped to assignment)

Assignment requirement	Where in this doc
Solution Design & Architecture + diagram + trade-offs	Section 1 (Figures 1–2)
Prompt Engineering: two real prompts + edge case	Section 2 (Prompts + noisy input example)
Sample Output + model/API used	Section 3 (Examples + model note)
Working Prototype (Python flow)	Section 4 (Core logic + pseudocode)
Retrieval Pipeline (10–20 incidents, top-K)	Section 5 (methods + hybrid)
Evaluation & Cost Awareness (metrics + monitoring)	Section 6 (Figure 3 + charts)

1. Solution Design & Architecture (How + Why)

We chose a Retrieval-Augmented Generation (RAG) architecture because the cost of hallucination in security operations is high. RAG grounds the LLM in known incident patterns and playbooks so outputs remain traceable and safer for analysts.

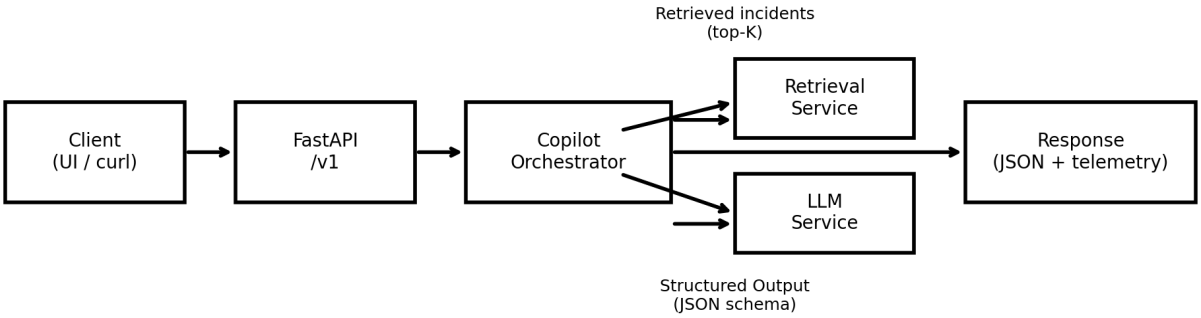

Figure 1 — End-to-end flow and component boundaries.

Why this architecture (decision-by-decision)

- **RAG vs LLM-only:** Reduces invented facts; enables citations to similar incidents.
- **Layered services:** API, orchestration, retrieval, and LLM can be tested and evolved independently.
- **Two endpoints (/retrieve and /analyze):** Cheap retrieval tuning + deterministic retrieval tests without LLM cost.
- **Structured outputs:** Schema-stable JSON for UI/tickets and reliable evaluation.

Trade-offs (and how we mitigate them)

- **Extra moving parts:** Requires an incident KB and retrieval tuning. Mitigation: baseline TF-IDF + optional FAISS/hybrid.
- **Latency:** Embeddings add cost/latency. Mitigation: cache embeddings; keep TF-IDF fallback.
- **Operational risk:** Misleading mitigation steps. Mitigation: environment-agnostic steps + open questions + grounding enforcement.

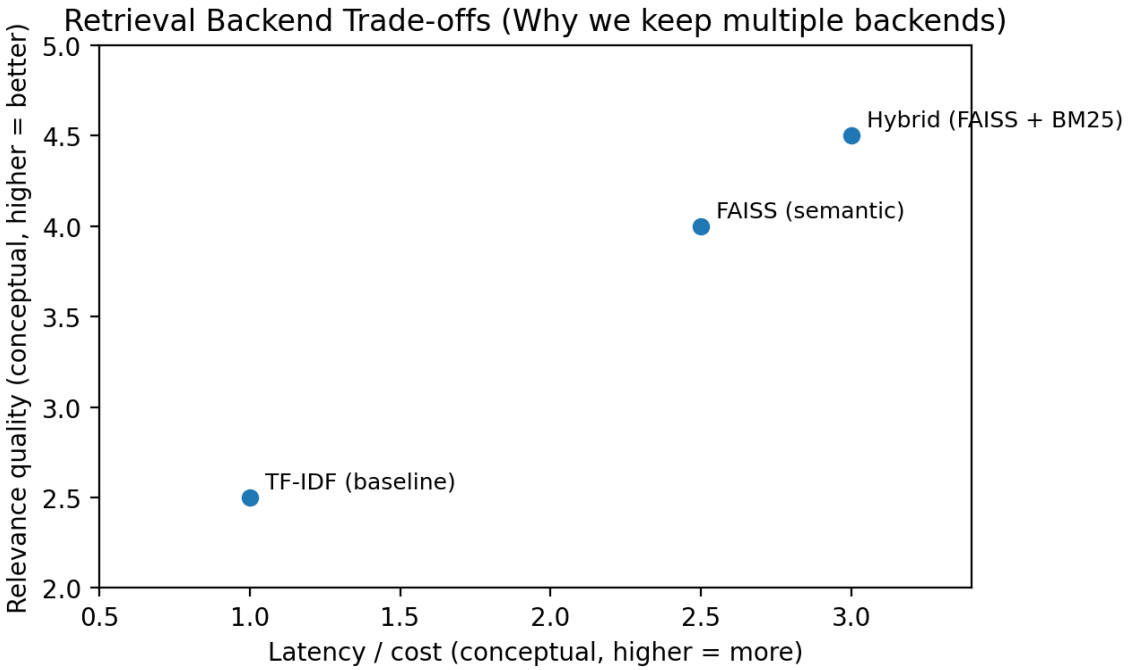

Figure 2 — Retrieval backend trade-offs (conceptual).

2. Prompt Engineering (How + Why)

Prompts are designed to prevent invention and keep outputs operational. We use a system prompt for safety rules and two task prompts for summary/classification and mitigation planning. JSON schema constraints reduce format errors.

System prompt (Why: prevent hallucinations)

```
You are a cybersecurity incident response copilot assisting SOC analysts.
Be precise, cautious, and operational. Do not invent facts.
If information is missing, state assumptions clearly and list open questions.
When referencing similar incidents, ONLY use the provided retrieved incidents and cite
their IDs (e.g., INC-003).
Output MUST be valid JSON and MUST follow the provided schema.
```

Prompt 1 – Summary/classification (How: bounded taxonomy)

```
Analyze the incident report and produce a compact, factual summary.
- Summarize only what is explicitly stated.
- Extract key indicators only if explicitly present.
- Assign suspected_category from: ransomware, phishing, malware, ddos,
account_compromise, dns_tunneling, unknown.
- If input is noisy/incomplete, keep summary minimal and add missing details to
open_questions.
Return ONLY JSON: summary, suspected_category, key_indicators, assumptions,
open_questions.
```

Prompt 2 – Mitigation plan (How: IR phases)

```
Given the incident report + retrieved similar incidents, create a mitigation plan.
Rules:
- Do NOT invent environment details.
- Prioritize containment -> eradication -> recovery -> prevention.
- Use retrieved incidents only as optional guidance and cite their IDs when used.
Return ONLY JSON: mitigation_plan {immediate, short_term, long_term, notes}.
```

Edge case example (noisy/incomplete input)

```
INCIDENT REPORT (noisy)
Something weird happened. Users complain files are slow. Maybe email issues? Not sure.
We saw 'login failed' messages but no details. Please investigate ASAP.
```

Expected behavior: produce a minimal summary, select suspected_category='unknown' if not supported by evidence, move uncertainty into assumptions and open_questions, and avoid suggesting environment-specific actions.

3. Sample Output (How)

Sample outputs were generated with **OpenAI gpt-4o-mini** using structured JSON outputs. The response includes telemetry (tokens, latency, estimated cost) for evaluation and budgeting.

Example input

```
INCIDENT REPORT
User reports they cannot open files; filenames changed to an unfamiliar extension.
Multiple hosts show rapid file modifications. A text file appears demanding payment.
IT noticed a spike in privileged logins overnight.
```

Example retrieval result

```
RETRIEVED INCIDENTS (top-2)
[
{"id": "INC-003", "title": "Ransomware - file extensions + ransom note", "score":
0.82},
{"id": "INC-011", "title": "Privileged account misuse preceding encryption", "score":
0.77}
]
```

Example output

```
STRUCTURED OUTPUT (excerpt)
{
"summary": "Multiple hosts show rapid file changes and inaccessible files with an
unknown extension; a ransom note is present; privileged login activity spiked
overnight.",
"suspected_category": "ransomware",
"key_indicators": ["unknown file extensions", "rapid file modifications", "ransom
note", "privileged login spike"],
"assumptions": ["Scope of impacted hosts is not fully known", "Initial access vector
is not confirmed"],
"open_questions": ["Which accounts performed the privileged logins?", "Were backups
reachable and untouched?", "Any known EDR alerts preceding encryption?"],
"mitigation_plan": {
"immediate": ["Isolate affected hosts", "Disable/reset suspicious privileged
accounts", "Preserve logs and artifacts"],
"short_term": ["Identify entry vector", "Eradicate persistence", "Restore from clean
backups"],
"long_term": ["Harden privileged access (MFA/PAM)", "Improve backup
immutability/testing", "Tune detections"],
"notes": ["Guidance referenced retrieved patterns (INC-003, INC-011)."]
}
}
```

4. Working Prototype (How)

Core flow implemented in Python: validate input → retrieve similar incidents → build grounded prompts → call LLM with schema → validate + enforce grounding → return JSON + telemetry.

```
PIPELINE (pseudocode)
1) validate(report_text)
2) matches = retrieval.search(report_text, top_k, use_hybrid)
3) prompt = build_prompt(report_text, matches)
4) llm_json = call_llm(model, prompt, json_schema)
5) validate_schema(llm_json)
6) enforce_grounding(llm_json.similar_incidents == matches)
7) return {analysis: llm_json, telemetry: tokens/latency/cost}
```

5. Retrieval Pipeline (How + Why)

We embed a small dummy set of incidents (10–20). For each incident, we build a document (title + description + tags + mitigation) and index it using either TF-IDF (baseline) or FAISS (semantic). Hybrid fusion is optional.

Methods used

- **TF-IDF + cosine:** fast, local baseline for iteration.
- **FAISS + embeddings:** better semantic recall when wording differs.
- **Hybrid search:** weighted fusion to reduce misses (semantic + keyword).
- **Top-K output:** return top 1–2 matches to keep prompts compact and reduce noise.

6. Evaluation & Cost Awareness (How + How Much + Why)

We evaluate both retrieval and generation quality and track telemetry to monitor drift, failures, and cost. Cost is estimated per run based on token usage and configurable per-1K token pricing.

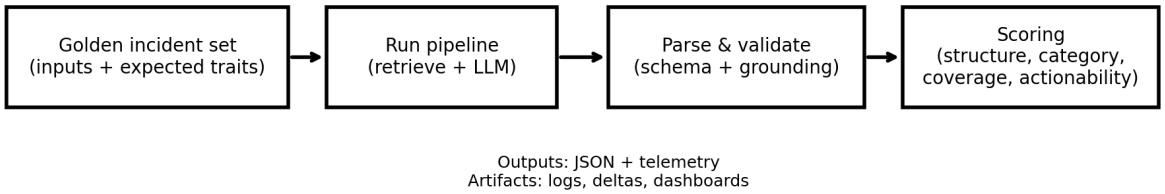

Figure 3 — Evaluation loop (golden set → pipeline → validation → scoring).

Cost estimation method (How much)

```
COST (method)
estimated_cost_usd = (input_tokens/1000)*price_in + (output_tokens/1000)*price_out
Example (from sample telemetry): input_tokens=733, output_tokens=478.
→ estimated_cost_usd = 0.733*price_in + 0.478*price_out
We include the computed value in the API response metadata.
```

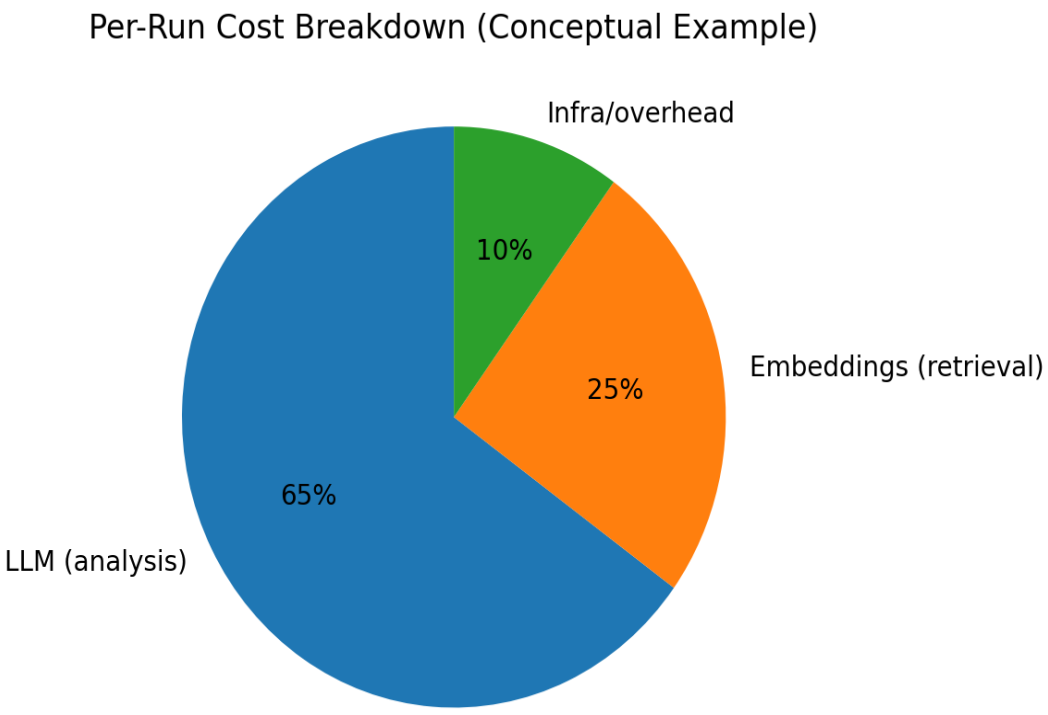

Figure 4 — Conceptual cost breakdown per run (LLM dominates; embeddings matter in FAISS mode).

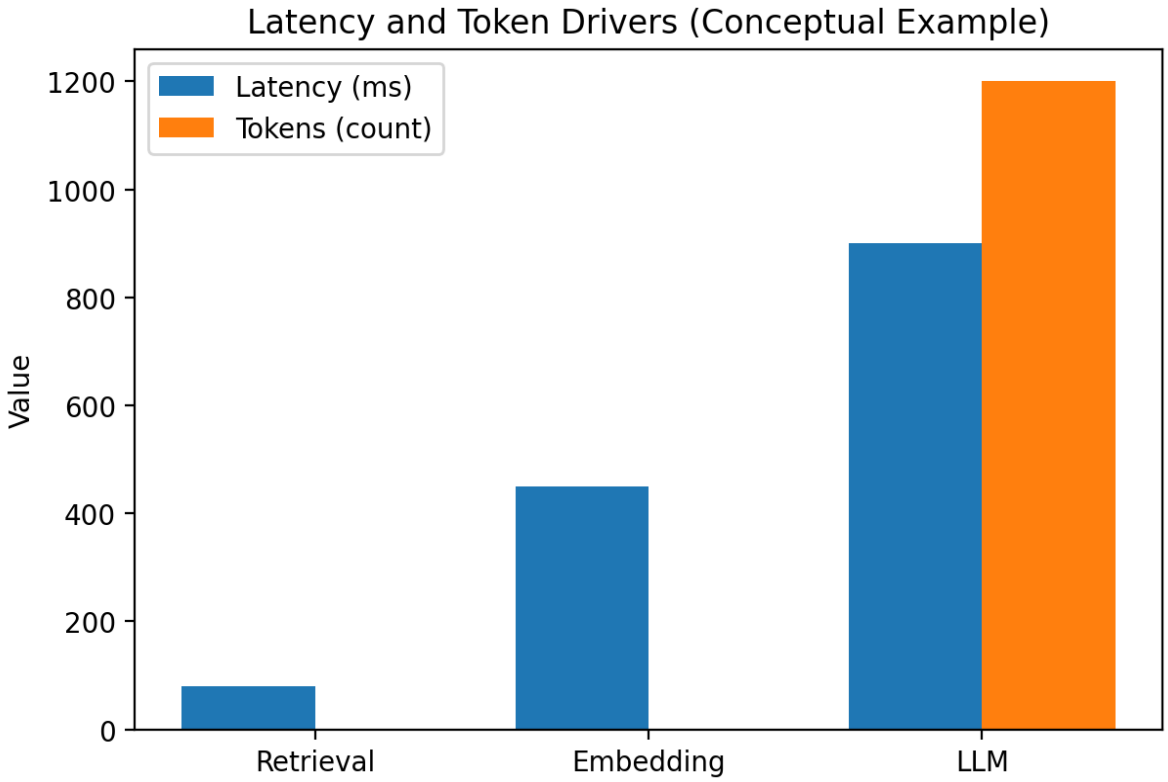

Figure 5 — Conceptual latency/token drivers (used to guide optimization work).

Monitoring (Why: production safety)

- **Latency:** p50/p95/p99; split embed latency vs search latency vs LLM latency.
- **Quality:** schema validity rate; grounding violations; category distribution drift.
- **Reliability:** API error rates; timeout rate; retries.
- **Cost:** tokens/run, cost/run, budget alerts; cache hit rate for embeddings.