ChatGPT

Luka sterowania ekonomią inferencji agentowej w Twojej architekturze SaaS+AI

Potrzeby informacyjne

Aby opisać lukę **mechanicznie** i zgodnie z Twoim repo (nie ogólnikami), muszę zestawić:
- jakie **pętle sterowania** masz już wdrożone w kodzie (gdzie jest pomiar, próg, akcja) – na poziomie CI/CD i runtime,
- gdzie w Twoim repo jest już język **„energia / próg / commit"** (HMK9D),
- gdzie brakuje analogicznej, egzekwowalnej pętli dla **kosztu AI w runtime** (tokeny/czas/koszt), mimo że masz już wzorzec eventów (sbom/scan/delta/gate),
- jak dokładnie przez tę lukę powstaje kaskada kosztowa i awaryjna.

Kontekst aktualności

To jest **dzisiejsza luka** (2026) dlatego, że rynek przeszedł z „LLM jako funkcja" na „LLM jako agent": jeden request generuje wiele kroków i zmienny koszt. Twoje repozytoria powstawały wcześniej, więc naturalnie są bardzo mocne w „sterowaniu strukturą" (supply chain, progi wdrożeniowe, rate limiting), a słabsze w „sterowaniu ekonomią agentowości" (telemetria kosztu per workflow/tenant i budżety).

Nazwa luki

Luka: „Brak bramki kosztowej dla AI w runtime"
(krócej: **AI Runtime Cost Gate Gap**)

W Twoim słowniku 9D to jest brak domknięcia mostu **„Semantyka–Energia"** w sensie operacyjnym: masz semantykę decyzji i masz ideę energii, ale nie masz **twardego progu i akcji** w runtime opartej o koszt pracy.

Dowód z repo: gdzie masz już pętle sterowania, a gdzie ich brakuje

Masz pełną pętlę sterowania w CI/CD (sbom)

W Jenkins pipeline w `sbom` masz kompletną strukturę:

1) **pomiar** (SBOM + scan + delta)
2) **próg** (`FAIL_ON` + policzone CRIT/HIGH)
3) **akcja** (`exit 10` = STOP)

To jest widoczne wprost:

- parametr progu:

```
choice(name: 'FAIL_ON', choices: ['none','critical','high'], description:
'Gate threshold')
```

- pomiar i decyzja:

```
CRIT=$(jq '[... select(.=="Critical")] | length' "$SCAN_JSON")
HIGH=$(jq '[... select(.=="High")] | length' "$SCAN_JSON")

if [ "$FAIL_ON" = "critical" ] && [ "$CRIT" -gt 0 ]; then DECISION="STOP"; fi
...
if [ "$DECISION" = "STOP" ]; then exit 10; fi
```

- oraz **event sourcing** do Elastic: `event_type: sbom_snapshot`, `scan`, `delta`, `gate`.
  To jest Twoja „maszyna dowodowa”: pomiar → historia → gate.
  (Źródło: `pipeline_one.pipeline` w Twoim repo `sbom` – treść pobrana z GitHub i zacytowana powyżej.)

Wniosek: potrafisz budować pętle sterowania i egzekwować „STOP”.

Masz pętlę „brzegową” w runtime (swarm)

W `swarm` wprowadzisz hamulce kaskad na bramie:

- **rate limit** (z fail-closed):

```
failure_mode_deny: true
timeout: 0.25s
```

- **circuit breaker/outlier detection**:

```
outlierDetection:
  consecutive5xxErrors: 5
  interval: 1s
  baseEjectionTime: 30s
  maxEjectionPercent: 100
```

(To są Twoje pliki `rate-limit.yaml` i `circuit-breaker.yaml`.)

Wniosek: potrafisz budować runtime-owe „pręty kontrolne” (hamulce kaskad przeciążeniowych).

Masz formalizm „energia + próg” (chunk-chunk), ale bez egzekucji kosztu

W `chunk-chunk/hmk9d_protocol.yaml` masz:

- zachowanie: `H(s) = g(F(s))`
- ryzyko: `R(F,g)`
- energia kroku: `E(Δ)` i `E_total`

- oraz **energy_guard**:

```
energy_guard:
  metric: "E"
  max_value: 0.8
```

Czyli w Twojej ontologii istnieje:
E jako koszt lokalny i globalny + próg maksymalny.

Wniosek: masz precyzyjny język, żeby zrobić bramkę kosztową.

Gdzie dokładnie jest luka i jak działa

Luka polega na tym, że:

- masz „gate" dla podatności (SBOM/scan),
- masz „gate" dla przeciążeń na brzegu (rate limit / circuit breaker),
- masz pojęcie „energii" i „progu",
- ale **nie masz analogicznej bramki runtime dla kosztu AI**: tokenów/czasu/kosztu per workflow/per tenant.

Czyli brakuje Ci odpowiednika:

`SCAN_JSON → CRIT/HIGH → FAIL_ON → STOP`

ale dla AI:

`AI_USAGE → tokens/time/tools → COST → BUDGET_POLICY → DEGRADE/DENY/HUMAN`

Mechanizm kaskady przez tę lukę (krok po kroku)

Poniżej mechanika jest „repo-spójna": pokazuje dokładnie, jak brak bramki kosztowej działa jak „dziura w pętli".

```
flowchart TB
  U[Użytkownik/tenant uruchamia agentowe zadanie] --> W[Workflow: wiele kroków tool-use/retry/retrieval]
  W --> T[Zużycie: tokeny/czas/narzędzia rośnie nieprzewidywalnie]
  T -->|brak telemetryki| Z[Brak mierzalnego E(Δ) w pieniądzu]
  Z -->|brak progu| P[Brak energy_guard w runtime (brak cap/degrade/deny)]
  P --> C[Koszt zmienny rośnie w ogonie (whales)]
  C --> M[Marża spada / billing surprise]
  M --> A[Ad hoc capy/pricing changes/degradacje]
  A --> R[Spadek zaufania + churn + presja na ROI]
  R --> U
```

To jest dokładnie „pętla w pętli” w Twoim rozumieniu, tylko że tutaj pętla nie jest amortyzowana przez bramkę – bo bramka istnieje dla SBOM i dla QPS, ale nie dla „E(Δ) w pieniądzu”.

Dlaczego rate limiting nie zamyka tej luki?

Bo Twoje rate limiting i circuit breaker sterują: - ruchem i awariami transportowymi (QPS/5xx), a luka jest o: - koszcie pracy „wewnątrz” (tokeny, iteracje, narzędzia, inference time).

Możesz mieć niskie QPS, ale bardzo drogie requesty (długi kontekst, dużo iteracji agentowych).
Czyli „brzeg” jest kontrolowany, a „energia” w środku nie.

Dodatkowy czynnik niwelujący asymetrię: wzorzec wewnętrzny w swarm

W Twoim `aggregator.py` każdy pakiet UDP jest obsługiwany w osobnym wątku:

```
threading.Thread(target=handle_message, args=(data, addr),
daemon=True).start()
```

a request do API idzie bez jawnego timeoutu:

```
response = requests.post(AGGREGATOR_API_URL, json=data_json)
```

To znaczy: nawet jeśli brzegowo masz hamulce, wewnątrz masz wzorzec, który przy burstach może eskalować liczbę wątków i koszty.
To jest analogiczny problem do agentowego AI: brak backpressure = brak sterowania energią kroku.

Jak lukę „domknąć” w języku Twojego repo

Ponieważ masz już wzorzec w `sbom` (eventy + gate + STOP), najczystsza naprawa jest taka:

Zrób „AI-gate” jako kopię „SBOM-gate”

1) **Event envelope** jak w pipeline (tylko typy inne):
- `event_type: ai_usage_snapshot`
- `event_type: ai_cost`
- `event_type: ai_delta`
- `event_type: ai_gate`

2) **Pomiar** (odpowiednik CRIT/HIGH):
- `tokens_in`, `tokens_out`, `latency`, `steps`, `tool_calls`
- `estimated_cost_usd` (albo „E(Δ) w pieniądzu”)
- `tenant_budget_remaining`

3) **Próg** (odpowiednik FAIL_ON):
- `BUDGET_MODE: none / soft_cap / hard_cap / degrade / deny`
- `MAX_COST_PER_WORKFLOW`
- `MAX_STEPS`, `MAX_TOOL_CALLS`

4) **Akcja**:
- degrade do tańszego modelu,
- deny (fail-closed),
- wymuś HITL,
- przerwij workflow (odpowiednik `exit 10`).

Zepnij to z HMK9D

W HMK9D masz energy_guard i most „Semantyka–Energia".
Domknięcie oznacza, że runtime liczy:

- `E(Δ)` jako koszt kroku,
- `E_total` jako koszt workflow, i **energy_guard** jest prawdziwą bramką, nie deklaracją.

Wtedy Twoja asymetria staje się pełna:
masz bramki dla supply chain, bramki dla transportu i bramki dla kosztu agentów.

Krótka definicja luki do przekazania „na zewnątrz"

> **Luka sterowania ekonomią agentowego AI** polega na tym, że system ma bramki bezpieczeństwa (SBOM) i bramki ruchu (rate limiting), ale nie ma bramki kosztowej dla pracy agentów (tokenów/iteracji/narzędzi), więc ogon kosztu może wywołać kaskadę marżowo-pricingową i wtórnie kaskadę reputacyjno-rynkową.

Jeśli chcesz, mogę zrobić dwie wersje tego samego opisu: - *wersję 8-linijkową do udostępnienia* (LinkedIn/X), - *wersję „dowód + testy + metryki"* z kryterium falsyfikacji i planem pomiarowym (jak w Twoich writeups).
