ChatGPT

Weryfikacja naukowa anty-wzorca wobec SaaS na kodzie DonkeyJJLove i jego zdolności amortyzacji kolapsu Cloud+AI

Zakres, materiał dowodowy i wykonane czynności

Badanie **rozpoczyna się od użycia włączonego konektora** entity["company","GitHub","code hosting platform"] oraz pracy **wyłącznie na sześciu wskazanych repozytoriach**: `DonkeyJJLove/sbom`, `DonkeyJJLove/writeups`, `DonkeyJJLove/chunk-chunk`, `DonkeyJJLove/swarm`, `DonkeyJJLove/ai_platform`, `DonkeyJJLove/HA2D` (bez użycia innych repo). W tej sesji wykonałem dwie klasy operacji: (a) identyfikacja i pobranie artefaktów „źródła prawdy" z repo (konektor GitHub → `get_repo`, `fetch` na plikach), (b) analiza statyczna: dokładna rekonstrukcja mechanizmów *metering → próg → akcja* oraz zaprojektowanie testów, które mogą te mechanizmy **sfalsyfikować**.

Z powodu ograniczeń narzędziowych środowiska (brak możliwości `git clone`/bezpośredniego uruchomienia Kubernetes/Istio w tym runtime), **nie wykonuję pełnych testów wykonawczych** (E2E/obciążeniowych) „na żywo" w mojej piaskownicy. Zamiast tego raport zawiera: (1) plan badawczy w rygorze falsyfikowalności, (2) wykonaną analizę kodu i konfiguracji, (3) dokładnie zdefiniowane testy oraz procedury ich uruchomienia w środowisku użytkownika/CI, wraz z kryteriami akceptacji/odrzucenia.

Dostępne artefakty i powód ich doboru (repo → pliki/ścieżki):

- `sbom`
- `lab/jenkins/pipeline_one.pipeline` – źródło prawdy o pętli: SBOM → scan → snapshot → delta → gate → ingest; kluczowe do weryfikacji mechanizmu „dowodu pochodzenia i zmiany" oraz bramki ryzyka.
- `docs/03_JENKINS_PIPELINE.md` – dokumentacja uruchomieniowa labu i lista zdarzeń indeksowanych jako dowód obserwowalności (event store) oraz replikowalności procesu.
- `swarm`
- `infrastructure/istio/policies/rate-limit.yaml` – definicja twardego progu ruchu (rate limit) jako zabezpieczenia przed kaskadą wolumenu.
- `infrastructure/istio/policies/circuit-breaker.yaml` – definicja wykrywania i odcinania outlierów (outlier detection) jako anti-cascade.
- `infrastructure/istio/policies/README.md` – pełna specyfikacja polityk (rate limiting, circuit breakers, fault injection, retries/timeouts, mirror traffic) oraz powiązanych metryk.
- `aggregator/aggregator.py`, `aggregator/mqtt_bridge/mqtt_bridge.py`, `aggregator-api/aggregator_api.py` – minimalna implementacja toru: ingestion → forwarding → persist, która pozwala projektować testy E2E i obciążeniowe (burst/sustained).
- `chunk-chunk`
- `hmk9d_protocol.yaml` – formalizacja kontraktu decyzyjnego (S, Σ, A, F, g, H, a*) oraz osi 9D i „mostów" jako operatorów sterujących; to rdzeń operacjonalizacji Twojej tezy „wioska zamiast fabryki": wieloosiowość + progi + gating.
- `ai_platform`

- `platform.md` – reguły mapowania „abstrakcyjnego woluminu 9D” (`/QV9D`) na strukturę katalogową platformy; istotne do sprawdzenia, czy governance jest „wymuszalne”, czy tylko opisowe.
- `LAT_GLX_PROJECT_MOSAIC.MD` – opis mozaiki repo i heurystyk mapowania warstw (INF/SEM/MAND) na artefakty (SPEC/STATE/METRICS/RITUAL/CI), co jest podstawą audytowalności i spójnej telemetrii.
- `HA2D`
- `context_protocol.md` – definicja *Context Memory Manager* (UUID + timestamp + payload + SHA256) i operacji STORE/RETRIEVE.
- `HUD_spec.md` oraz `readme.md` – definicja warstwy introspekcyjnej (HUD) i „pamięci synaptycznej” jako artefaktów sterowania i replikowalności procesu.
- `writeups`
- `README.MD` – indeks materiałów badawczych i narzędzi epistemicznych (P0–P3, mikrokod, protokoły kontekstu) oraz obszarów ryzyka (fire sale, koszt błędu, HITL).
- `protokoly_kontekstu_chunk-chunk_facebook_case.md` – formalny zapis „protokołu kontekstu” jako dynamiki stan→decyzja (Fθ, G), wraz z explicytną sekcją „dowód (warunkowy) i falsyfikacja”, co jest bezpośrednio użyteczne jako wzorzec metodologiczny.

W tle badania leży teza o kolapsie Cloud+AI jako konsekwencji sprzężeń: koszt energii/compute rośnie szybciej niż tradycyjne liczniki wartości, a infrastruktura energetyczna ma dłuższe czasy realizacji niż cykle wdrożeniowe data center. [1]

Pytania badawcze i hipotezy falsyfikowalne

Aby metrycznie stwierdzić, czy Twoje podejście **amortyzuje** (zmniejsza amplitudę, częstotliwość i koszt) problemów, które doprowadziły do zapaści ekonomicznej Cloud+AI, trzeba rozstrzygnąć następujące pytania (3–6 kluczowych):

Czy metering przypisuje koszt do jednostki pracy, a nie do użytkownika?
W AI-SaaS „wolumen” staje się funkcją iteracji/agentów, a nie seatów. Jeśli system nie potrafi policzyć kosztu na „work unit”, pricing nie jest regulatorem, tylko syreną alarmową.

Czy gating (CI/CD i runtime) realnie przerywa kaskady?
W systemach progowych mały impuls może uruchamiać duże kaskady – „kiedy sieć jest gęsta, a progi reakcji niskie”. [2]
Sprawdzamy, czy Twoje progi (SBOM gate, rate limit, circuit breaker) zmniejszają kaskady incydentów/awarii i kosztów.

Czy kontekst jest replikowalny, wersjonowalny i dowodowy?
Czy system potrafi: (a) odtworzyć decyzję, (b) wskazać różnicę kontekstu („delta kontekstu”), (c) wykazać integralność (hash), czyli spełnić wymóg eksperymentu replikowalnego.

Czy architektura jest wieloosiowa (sens/energia/ryzyko/czas), a nie monometryczna?
Jeśli decyzje są podejmowane w jednej osi (np. time-to-market), system prędzej czy później przekroczy próg złożoności i eksploduje. W Twoim podejściu te osie są jawne w protokole 9D.

Czy podejście zmniejsza EV błędu i TCO w scenariuszach cenowych/energetycznych?
W gospodarce mocy obliczeniowej krytyczny jest nie tylko koszt średni, lecz również ogon ryzyka (ciężkoogonowe incydenty), a to wymaga symulacji i testów statystycznych.

Na tej podstawie definiuję hipotezy w reżimie falsyfikowalności:

H1 (amortyzacja kaskad): Wprowadzenie pętli „pomiar → próg → akcja” (SBOM/delta/gate oraz rate-limit/circuit breaker) obniża (i) częstość kaskadowych awarii, (ii) czas regeneracji, (iii) koszt incydentów, w porównaniu do wariantu „bez progów”.

H2 (kontekst jako dowód): Zastosowanie protokołu kontekstu (UUID+SHA256) oraz formalnych osi (9D) zwiększa replikowalność wyników i pozwala wyjaśnić różnice wyników przez różnice warunków (delta), a nie przez narrację.

H3 (licznik wartości): Podejście umożliwia przejście z „seat economics” do „work-unit economics” przez metering kosztu i egzekwowalne progi, a więc ogranicza przypadki, w których przychód jest płaski, a koszt zmienny i opóźniony.

H0 (hipoteza zerowa): Mechanizmy są deklaratywne i nie przekładają się na mierzalną poprawę metryk (kaskady, koszt/jednostka pracy, EV błędu), albo ich koszt operacyjny przewyższa zysk amortyzacyjny.

Operacjonalizacja: metryki i modele kosztu

Metryki techniczne

Metryki techniczne muszą „widzieć” pętle agentowe i integracje (czyli to, co w AI generuje koszt i kaskady), a nie tylko klasyczny ruch HTTP:

- Latencja p50/p95/p99 (per endpoint, per service, per try; osobno dla retry).
- Throughput (RPS/QPS) oraz „burst survival time” (czas do osiągnięcia degradacji).
- Error budget: odsetek 5xx, odsetek 429 (rate limit), odsetek timeouts, liczba ejection w outlier detection.
- Zużycie zasobów: CPU, RAM, I/O, połączenia (conn pool), kolejki i backlog.

Metryki operacyjne

- MTTR i czas powrotu do „stanu stabilnego” po obciążeniu.
- Wskaźnik kaskadowości: procent incydentów, które propagują się downstream (np. z ingestion do API i DB).
- Jakość bramek: false positives / false negatives dla gate (SBOM) oraz dla progów sieciowych.

Metryki ekonomiczne

Najważniejsze: koszt na jednostkę pracy (*work unit*), a nie na użytkownika.

Definicje operacyjne:

- Work unit (WU) = minimalna porcja pracy, która generuje koszt compute: np. „jedna iteracja agentowa”, „jedno wywołanie narzędzia”, „jedno przetworzenie wiadomości MQTT/UDP + zapis do DB”.
- Cost per WU:

$$C_{WU} = \frac{C_{compute} + C_{storage} + C_{egress} + C_{ops}}{N_{WU}}$$

gdzie C_{ops} obejmuje koszt weryfikacji, obserwowalności, on-call, regresji i audytu.

- EV(ryzyka błędu) – oczekiwana strata z błędów:

$$EV_{err} = \sum_k p_k \cdot L_k$$

gdzie p_k to prawdopodobieństwo klasy błędu (np. incydent kaskadowy, regresja bezpieczeństwa, dryf kontekstu), a L_k to koszt (czas, pieniądz, utrata danych/klienta, reputacja).

W kontekście Cloud+AI realny stresor ekonomiczny jest sprzężony z energią: rosnące zapotrzebowanie energetyczne data center jest według entity["organization","International Energy Agency","energy policy agency"] w scenariuszu bazowym prognozowane jako „ponad podwojenie" globalnej konsumpcji do ok. 945 TWh do 2030, a cykl wdrożeniowy data center bywa krótszy niż cykl rozbudowy systemu energetycznego. [1]
To wzmacnia sens hipotezy: nawet jeśli architektura jest „teoretycznie optymalna", brak progów i meteringu w warunkach presji kosztu energii prowadzi do kaskad.

Analiza repozytoriów: mechanizmy amortyzacji i luki

W tej sekcji nie „opisuję idei", tylko mapuję: **mechanizm → jaki problem SaaS rozwiązuje → jak to falsyfikować → gdzie są luki**.

`sbom` **– amortyzacja przez dowód pochodzenia, różnicę i bramkę**
Pipeline wprost implementuje pętlę: generacja SBOM (CycloneDX JSON), skan podatności, snapshot komponentów, pobranie poprzedniego snapshotu, obliczenie delta, decyzja gate (GO/STOP), a następnie ingest zdarzeń do event store (Elastic) wraz z tożsamością AID (env/app_id/repo/vcs_ref/version). To jest praktyczna implementacja „kontekst jako artefakt", ale w domenie supply chain.
Naukowa wartość: w świecie ciężkoogonowych ryzyk (rzadkie, kosztowne incydenty) brak takiej pętli oznacza, że system działa „aż do dnia", kiedy przestaje. Z jawnie zdefiniowaną deltą możesz projektować testy typu: *czy delta koreluje z ryzykiem* oraz *czy gate realnie zatrzymuje regresję*.
Luka do domknięcia ekonomicznego: gate jest zdefiniowany w kategoriach severity (critical/high) i decyzji STOP/GO, ale bez bezpośredniego przeliczenia na koszt/jednostkę pracy lub na EV błędu. To nie błąd; to miejsce na metrykę „ile kosztuje złe GO oraz ile kosztuje fałszywe STOP".

`swarm` **– amortyzacja przez runtime-owe progi i topologię integracji**
Tu masz to, czego brakowało SaaS-om w modelu „unlimited": progi w service mesh (rate limiting) oraz odcinanie outlierów (circuit breaker/outlier detection), a dokumentacja dopina jeszcze fault injection, retries/timeouts i mirror traffic (co pozwala robić quasi-A/B na ruchu).
Z punktu widzenia kaskad: to są mechanizmy, które mają podnieść progi reakcji i ograniczyć propagację błędu. Modele progowe zachowań zbiorowych pokazują, że rozkład progów determinuje, czy mały impuls uruchomi globalną reakcję. [2]
Luka: progi są skonfigurowane na poziomie ruchu i błędów (5xx, timeouts), ale w AI często koszt jest funkcją agentowych pętli i narzędzi, więc kontrolna metryka progu musi zostać skalibrowana do „kosztu pracy", a nie tylko do RPS.

`HA2D` **– amortyzacja przez protokół kontekstu jako dowód i HUD jako interfejs sterowania**
`context_protocol.md` wprowadza rekord kontekstowy (UUID, timestamp, payload, SHA256) oraz operacje STORE/RETRIEVE/LATEST, czyli minimalny reżim: *co było*, *kiedy było*, *czy nienaruszone*, *jak przywołać*. HUD opisuje, jak tę integralność i „trend" pokazywać operacyjnie jako system sterowania (nie

jako log).
To jest kluczowe dla Twojej tezy o „wiosce": w systemie zasobowo zamkniętym musisz widzieć progi zanim spalisz zasób (energia, uwaga, koszt błędu).

`chunk-chunk` **– amortyzacja przez formalizację osi i progów jako część kontraktu decyzyjnego**
`hmk9d_protocol.yaml` domyka reżim naukowy: definiuje przestrzeń stanu, kompresor, politykę, ryzyko (expected loss) i model energii lokalnej E(Δ) oraz 9 osi, w których można mierzyć proces (czas, sens, relacja, energia poznawcza itd.). To jest mechanizm anty-SaaS, bo SaaS monometryczny „działa aż nie działa". Tu z definicji nie da się udawać, że energia i ryzyko są „obok".
Luka: dokument jest bogaty semantycznie, ale aby był „wymuszalny" jak `sbom` i `swarm`, potrzebuje spięcia z CI/runtime (czyli: generować metryki i gate'y automatycznie, nie tylko opisywać).

`ai_platform` **– amortyzacja przez mapowanie i governance wielorepowej platformy**
`platform.md` i `LAT_GLX_PROJECT_MOSAIC.MD` próbują rozwiązać problem integracji i audytu w systemie wielorepowym: każdemu modułowi przypisujesz współrzędne w woluminie 9D oraz typ artefaktu (SPEC/STATE/METRICS/RITUAL/CI). To jest operacyjnie mocne, bo pozwala raportować metryki i koszty nie „po usługach", tylko po funkcjach i warstwach.
Luka: w `platform.md` występuje jawne miejsce „DO ZAPROJEKTOWANIA" w obszarze deterministycznego wyliczania identyfikatora, co w reżimie dowodowym jest nie do pominięcia: identyfikatory muszą być deterministyczne i odporne na kolizje, inaczej nie ma replikowalności.

`writeups` **– amortyzacja przez metodykę falsyfikacji i narzędzia epistemiczne**
W odróżnieniu od pozostałych repo, to jest repo metody i badań: zawiera explicytną strukturę „dowód warunkowy i falsyfikacja" oraz formalizuje protokół kontekstu jako funkcje przejścia i decyzji. To jest użyteczne jako wzorzec: jak budować eksperymenty A/B dla systemów, w których część mechanizmów jest czarną skrzynką.

Tabela porównawcza: SaaS-owy model vs anty-wzorzec z Twoich repo

Wymiar	Typowy SaaS (w warunkach AI/ Cloud)	Anty-wzorzec w Twoich repo
Metering	Seat/plan jako licznik; koszt compute często ukryty lub opóźniony	„Work-unit thinking" wymuszane przez zdarzenia, delty, kontekst i progi; koszt ma wejść do modelu decyzyjnego
Gating	„Unlimited / fair use" + manualny on-call	Automatyczne bramki: SBOM gate (CI) oraz rate-limit/circuit breaker (runtime)
Kontekst	Logi / pamięć w głowach; mała replikowalność	Context protocol (UUID+hash), delta-język i formalne osie; HUD jako warstwa sterowania
Przenaszalność	Integracje jako „glue code"; topologia rośnie organicznie	Świadome mapowanie warstw i artefaktów (platform/QV9D) + service mesh policies
Koszt	Koszt rośnie nieliniowo przy agentowych pętlach; pricing pęka	Progi + obserwowalność mają obcinać ogon ryzyka; energia i koszt mają być jawne

Projekt eksperymentów i procedury falsyfikacji

Poniżej plan testów w czterech klasach: jednostkowe, integracyjne, obciążeniowe, ekonomiczne. Każdy test ma: wejścia, procedurę, metryki, kryteria sukces/porażka, środowisko.

Testy jednostkowe

Cel: dowieść poprawności lokalnych mechanizmów (bez mylenia ze skutecznością systemową).

- `sbom`: testy deterministyczności delta
  Wejście: dwa snapshoty (lista komponentów) o kontrolowanej różnicy.
  Procedura: uruchomienie etapu DELTA, walidacja liczników added/removed i list.
  Metryki wyjściowe: zgodność added/removed, stabilność sortowania, brak niedeterministycznych wyników.
  Kryterium sukcesu: delta identyczna dla identycznych wejść; brak „dryfu" (0 różnic) przy braku zmian.
  Kryterium porażki (falsyfikacja): delta zmienia się mimo identycznego snapshotu.

- `swarm`: testy konfiguracji polityk
  Wejście: pliki YAML polityk.
  Procedura: walidacja schematów (istioctl analyze w środowisku K8s), uruchomienie test ruchu.
  Metryki: poprawność zastosowania, brak konfliktów zasobów.
  Porażka: polityki nie są stosowane lub są ignorowane.

- `HA2D`: test integralności kontekstu
  Wejście: payload JSON.
  Procedura: STORE → obliczenie SHA256 → RETRIEVE → ponowne obliczenie SHA256 i porównanie.
  Metryki: zgodność hash, deterministyczność serializacji.
  Porażka: hash niezgodny (naruszenie lub różna kanonizacja JSON).

Testy integracyjne (end-to-end)

Cel: dowieść, że pętle *pomiar* → *próg* → *akcja* działają jako układ.

- `swarm` E2E: ingestion → API → DB
  Scenariusz: generator wiadomości UDP/MQTT wysyła poprawne i błędne payloady w zadanym tempie; API zapisuje do DB.
  Metryki: p95/p99 latencji, odsetek 5xx, odsetek 400 (walidacja), backlog, MTTR po przeciążeniu.
  Kryterium sukcesu: under load system degraduje się kontrolowanie (rate-limit/429 zamiast kaskady 5xx), po ustaniu obciążenia wraca w T<MTTR_max.
  Kryterium porażki: cascading failure (np. lawinowy wzrost 5xx, brak odzysku).

- `sbom` E2E: pipeline → event store
  Scenariusz: dwa kolejne buildy: benign change vs ryzykowna zmiana zależności (symulowana).
  Metryki: gate decision, false positive rate, false negative rate, koszt pipeline (czas + zasoby).
  Kryterium sukcesu: gate STOP dla ryzykownego przypadku w ustalonym progu, GO dla benign; w ES widoczne zdarzenia w kolejności wspierającej delta.

Testy obciążeniowe

Cel: symulować agentową eksploatację (burst + sustained) i sprawdzić, czy progi amortyzują.

- Workload A: burst (nagłe skoki)
  Wejście: 10×–100× baseline RPS przez krótki czas.
  Oczekiwany efekt: rate-limit blokuje nadmiar (429), a circuit breaker odcina outliery.
  Metryka: „kaskadowość" (liczba downstream errors), czas do stabilizacji.

- Workload B: sustained (długotrwały nacisk)
  Wejście: 2×–5× baseline przez długi czas.
  Oczekiwany efekt: stabilny throughput w degradacji, brak narastającego backlogu, brak „rozjazdu kosztu".
  Kryterium porażki: dryf w czasie do awarii (system „trzyma" chwilę i pęka).

Eksperymenty ekonomiczne

Cel: przełożyć metryki techniczne na hipotezę o amortyzacji kosztu i ryzyka.

- Symulacja cost per work unit
  Wejście: logi żądań, liczba WU, eksport billingowy (compute, storage, egress), koszt pracy operacyjnej (on-call).
  Wynik: C_{WU} per scenariusz, per dzień, per klient.

- Monte Carlo scenariuszy cenowo-energetycznych
  Uzasadnienie: IEA pokazuje istotne niepewności (scenariusze bazowe/sensitivity) i dynamikę wzrostu, a to oznacza ryzyko zmiany kosztu energii/compute w horyzoncie wdrożeniowym. [1]
  Procedura: losujesz (i) koszt compute, (ii) wolumen WU (agentowość), (iii) prawdopodobieństwo incydentu kaskadowego, a następnie liczysz rozkład EV_{err} oraz C_{WU} dla wariantów „z progami" i „bez progów".
  Kryterium sukcesu H1/H3: redukcja ogona rozkładu kosztów (np. 95-percentyl C_{WU}) oraz redukcja EV_{err} ponad koszt wdrożenia mechanizmów.

Procedury falsyfikacji i statystyka

W reżimie naukowym falsyfikacja jest równie ważna jak „sukces".

- Dla H1: test różnic średnich i różnic percentyli (p95/p99, MTTR, liczba downstream failures) między wariantami A/B.
  Minimalny poziom istotności: α=0.05; w praktyce dla metryk ciężkoogonowych rekomendowane raportowanie również mediany i percentyli (nie tylko średniej).
- Dla H2: test replikowalności
  Kryterium obalenia: brak zdolności do wskazania delta kontekstu przy zmianie wyniku lub brak zgodności hash.
- Dla H3: regresja kosztu względem WU
  Kryterium obalenia: brak stabilizacji C_{WU} lub brak spadku „incydentów kosztowych" mimo progów; to oznacza, że progi są źle skalibrowane do realnego kosztu.

Plan wdrożenia, raportowanie, ryzyka i ograniczenia

Plan A/B i bezpieczeństwo (rollback)

A/B w środowisku testowym/produkcyjnym wymaga bezpieczników:

- Feature flagi dla progów (rate limit, breaker, retry policy) z natychmiastowym rollback.
- Mirror traffic (jeśli dostępne) jako quasi-eksperyment: nowa konfiguracja dostaje kopię ruchu, ale nie wpływa na odpowiedź.
- Stop-conditions: automatyczne wycofanie po przekroczeniu error budget lub degradacji p99.

Szablony raportów wynikowych i interpretacja

Każdy run powinien generować raport w tym schemacie:

- Konfiguracja (hash/wersja), scenariusz obciążenia, czas, środowisko.
- Metryki techniczne (latencje/429/5xx), operacyjne (MTTR/kaskadowość), ekonomiczne (C_{WU}, EV_{err}).
- Wniosek falsyfikacyjny: które hipotezy przeszły, które zostały obalone, w jakich warunkach.
- Rekomendacja: podnieść/obniżyć progi, zmienić metrykę sterującą progiem, wyłączyć funkcję, zmienić topologię integracji.

Harmonogram działania

- Dzień 1–2: „inventory + instrumentacja”
  Ustalenie definicji WU, metryk, eksport billingowy, pipeline raportowy.
- Dzień 3–5: testy jednostkowe i integracyjne offline (CI).
- Tydzień 2: obciążeniowe + kalibracja progów + pierwsze Monte Carlo.
- Tydzień 3: A/B w środowisku kontrolowanym z rollback + raport końcowy.

Wymagania środowiskowe i budżetowe

- Środowisko sieciowe: klaster testowy (może być lokalny) do weryfikacji polityk mesh oraz do powtarzalnych obciążeń.
- Storage/logging: retencja logów i metryk dla porównań A/B.
- Koszt GPU: w tym badaniu nie jest główny, chyba że dołączysz realne inferencje agentowe; wówczas trzeba budżetować „workload AI” osobno.

Energetyczne ograniczenia przypominają, dlaczego testy kosztu są obowiązkowe: wzrost zapotrzebowania data center jest prognozowany jako szybki, a czasy rozbudowy infrastruktury energetycznej są długie; to rodzi ryzyko bottlenecków i zmienności cen w horyzoncie produktowym. [1]

Ryzyka i ograniczenia badania oraz minimalizacja

- Brak dostępu do realnych danych billingowych → minimalizacja: syntetyczny billing + zakresy scenariuszowe, a następnie walidacja na ograniczonej próbce produkcyjnej.
- Koszt testów (czas i inżynieria) → minimalizacja: zaczynać od testów, które najszybciej falsyfikują H1/H2 (progi i replikowalność).
- Prywatność/bezpieczeństwo danych → minimalizacja: anonimizacja payloadów, testy na danych syntetycznych.
- Ograniczenia symulacji Monte Carlo (model błędu może być źle skalibrowany) → minimalizacja: kalibracja p_k i L_k z realnych incydentów, iteracyjne uaktualnianie.

Wnioski falsyfikowalne, które będzie można postawić po testach

Po wykonaniu planu z tego raportu będziesz w stanie powiedzieć (w sensie naukowym):

- czy mechanizmy `sbom` oraz `swarm` **realnie obcinają kaskady** (H1) czy są tylko „ładną deklaracją”,
- czy `HA2D` + `chunk-chunk` faktycznie dają **replikowalność i dowodowość kontekstu** (H2),
- czy spina się przejście z seat → work unit w metrykach kosztowych (H3),
- oraz gdzie dokładnie leży granica: *czy to progi są złe, czy metryka progu nie odpowiada kosztowi, czy topologia integracji generuje ogon ryzyka*.

W efekcie dostaniesz nie „opinię o tezie”, tylko wynik w stylu: **H1/H2/H3 przyjęta/obalona w warunkach X, z marginesem błędu Y, wraz z rekomendacją zmian progów i liczników**.

3

---

1 https://www.iea.org/reports/energy-and-ai/energy-demand-from-ai
https://www.iea.org/reports/energy-and-ai/energy-demand-from-ai

2 3 https://sociology.stanford.edu/publications/threshold-models-collective-behavior
https://sociology.stanford.edu/publications/threshold-models-collective-behavior