2026 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SANTA CLARA, CA, USA
MARCH 2 - 5, 2026

Dhenara Inc

Passive Token Accounting for Cost-Aware LLM Assistance In continuous SoC verification workflows

Ajith Jose

Agenda

- Why token cost becomes an operational risk
- PTA: what it is, and what “passive” means
- 3-layer design (client → execution → regression snapshot)
- Policies: budgets, soft caps, model selection
- Results, limitations, and future work

Motivation: LLMs are everywhere; cost is opaque

- LLMs in verification: summarization, clustering, spec Q&A, debug planning
- Regressions are **continuous** → spend compounds
- Agentic flows add retries, tool calls, multi-turn loops
- Traditional EDA metrics don’t natively track “LLM cost”

The questions teams ask (and can’t answer)

- What did triage cost last week?
- Which tests / phases burn the most tokens?
- Did a heuristics change double our bill?
- Can we cap spend without turning the feature off?

Design requirements

- **Transparent**: works across providers, minimal provider-specific code
- **Low overhead**: no measurable slowdown; local I/O only
- **Actionable**: per-regression + per-agent attribution
- **Enforceable**: budgets, soft caps, model downgrade policies

PTA in one sentence

Passive Token Accounting (PTA)

- Extract usage from model responses
- Normalize usage → cost across providers
- Aggregate into append-friendly snapshots
- Expose via lightweight API/UI

Where PTA lives: 3 layers

1) **LLM client layer** (open source: Dhenara AI `*dhenara-ai*`)
2) **Execution aggregation layer** (agent runtime boundary)
3) **Regression snapshot layer** (artifact SSOT)

Layer 1: provider-agnostic usage schema (Dhenara AI)

In Dhenara AI (MIT, open source):

- *ChatResponseUsage*:
  - *prompt_tokens*, *completion_tokens*, *total_tokens*
  - optional *reasoning_tokens*
- *UsageCharge*:
  - normalized *cost*
  - optional *charge* (cost × margin)

Normalizing cost across providers

- Input/output rates vary per provider + per model
- Normalize via a cost table:
  - *cost = prompt_tokens × input_rate + completion_tokens × output_rate*
- Optional *charge* via *cost_multiplier_percentage*

Streaming: accounting without losing correctness

- Streaming responses can emit partial usage deltas
- PTA approach:
  - accumulate deltas when present
  - finalize with end-of-stream usage when available
  - prefer conservative aggregation

Layer 2: execution-level aggregation (agent boundary)

Why aggregation matters:

- Multi-step flows: retries, tools, branching, subloops
- Need attribution per execution unit:
  - per agent
  - per phase
  - per run

Layer 3: regression-scoped snapshot artifact

Append-friendly snapshot (e.g., `costs.json`):

- `totals`: cost/charge + token totals
- `per_agent`: cost/charge + tokens + run counts
- `runs[]`: execution entries with timestamps + sanitized metadata

From artifact to UI: lightweight gateway

- Gateway reads snapshots → serves UI
- No DB required for evaluation
- UI answers immediately:
  - total cost/tokens
  - per-agent breakdown
  - run list with drill-down links

Policy controls: budgets and soft caps

Budget-aware knobs:

- per-phase limits: *max_*_tokens*, *max_*_cost*, time/depth caps
- model selection per phase
- soft cap behaviors:
  - downgrade model
  - reduce exploration
  - stop expensive subloops

Evaluation: what we observed

Practical outcomes:

- Negligible overhead (LLM dominates runtime)
- Cost visibility exposes heavy hitters
- Policy tweaks reduce spend without hurting output quality

Limitations

- Provider reporting gaps (failures, streaming)
- Local models may not emit usage
- Enforcement scope is policy-driven (not magic)

Future work

- Make accounting more complete (failures + streaming gaps)
- Standard exporters: Prometheus / OpenTelemetry
- Standard benchmarks for cost-aware LLM workflows

Dhenara AI (open source): why it exists

Dhenara AI (`dhenara-ai`)

- MIT-licensed, provider-agnostic LLM client
- Typed schemas (Pydantic)
- Streaming + async
- Usage/cost/charge data as first-class output

Where this fits in dVI™ (high-level)

dVI™ (Dhenara Verification Intelligence)

- Silicon regression **autopilot**
- PTA enables safe, predictable operation:
  - cost visibility per regression and per phase
  - budget-aware autopilot attempts (stop or downgrade safely)
  - audit trail that leadership can trust

Takeaways + closing

Takeaways
- Cost becomes a system problem in continuous regressions
- PTA gives cost observability without changing test flows
- Provider-agnostic schemas enable portability and policy control

Thank you (DVCon US 2026)
- Ajith Jose — Dhenara Inc.
- ajith@dhenara.com