

AEGIS KV-Cache Lab: An Evidence-Aware Architecture, Prototype Evaluation, and Integration Roadmap

Aaron Singh

Abstract—This paper presents AEGIS KV-Cache Lab, a project in a twenty-project AI infrastructure portfolio. The current repository is an active prototype with implemented behavior, automated correctness tests, and explicit boundaries around unverified hardware or production claims. We describe the system model, current implementation, goals, and evaluation method, then propose five integrations that advance the project toward reproducible system-level validation. A local audit executed 5 project tests successfully. No accelerator, deployment, or performance conclusion is inferred unless a corresponding committed benchmark or profiler artifact exists.

Index Terms—AEGIS, AI infrastructure, reproducibility, prototype validation, systems evaluation, hardware–software co-design

I. INTRODUCTION

AEGIS KV-Cache Lab (P05) addresses a bounded problem within the portfolio’s track-02-llm-serving track. Its declared status is *Active Prototype* [1]. The project validates its central mechanism before adding device-specific acceleration, production orchestration, or real-world hardware. This ordering matters because optimization without a trusted reference can make incorrect behavior appear successful.

The project goals are to establish deterministic behavior, encode correctness as tests, create machine-readable evidence, identify limiting resources through measurement, and integrate results with adjacent portfolio systems without losing provenance. The repository contains one paper named exactly after its singular folder: `P05-aegis-kv-cache-lab.tex`.

II. TECHNICAL CONTEXT

Language-model serving couples execution with cache management, request scheduling, and latency objectives. A simulator can validate policy logic, but only a real model runtime and measured workload can support performance claims. The portfolio benchmark standard requires environment, workload, method, metric, artifact, reproduction, and limitation fields; unknown values remain explicitly unmeasured [3].

III. SYSTEM MODEL AND ARCHITECTURE

The prototype is organized around the following domain model:

$$p(r) = \left\lceil \frac{t_r}{T_p} \right\rceil, \quad \sum_r p(r) \leq P_{total} \quad (1)$$

The equation is a design and test abstraction rather than a claimed empirical law. It supports invariants and expected-value checks while later implementations replace synthetic inputs with representative workloads or devices.

The software architecture contains an input/configuration layer, a deterministic core, validation and evidence output, and a local visualization. The principal inspected source artifacts are `python/kv_cache_simulator.py`. Unsupported real-world conditions are surfaced as limitations rather than silently simulated.

IV. DETAILED SCRIPT OPERATION AND RATIONALE

The simulator reads a JSON workload and models a fixed pool of KV-cache pages. It rounds token demands to pages, allocates and releases ownership, rejects exhaustion safely, reuses freed pages deterministically, and checks that no page is multiply owned.

The execution path is:

- 1) Load page count, page size, and request operations from JSON.
- 2) Round each token demand upward to whole pages.
- 3) Allocate free pages or reject the request atomically on exhaustion.
- 4) Release pages and preserve deterministic reuse order.
- 5) Check ownership/free-list invariants after every operation and emit a trace.

V. IMPLEMENTED PROTOTYPE

The metadata and source audit found these completed features [2]:

- Python paged KV-cache allocator
- Allocation, exhaustion, release, and deterministic page reuse
- Internal page ownership invariants and unit tests
- JSON workload runner and trace output

`python3 -m unittest discover -s tests -p 'test_*.py'` completed successfully with 5 tests. The inspected test artifacts are `tests/unit/test_kv_cache.py`. This is evidence of local correctness for encoded cases, not production scale or hardware performance.

VI. TESTING METHODOLOGY AND OBSERVED RESULTS

Testing uses Python’s standard `unittest` discovery and exercises the public behavior of the reference implementation. The audit reran the suite from the project folder with `python3 -m unittest discover -s tests -p 'test_*.py'`. All 5 discovered tests passed. The result establishes correctness only for the encoded

TABLE I
IMPLEMENTATION ARTIFACTS AND WHY THEY EXIST

Artifact	Observed responsibility	Engineering rationale
python/kv_cache_simulator.py	PagedKVCache, run_workload, main	Implements the inspectable, unit-tested project core.
scripts/reproduce.sh	Fixed test and demonstration entry point	Gives another developer one command for local reproduction.
PROJECT.yaml	and Status, completed work, planned work, and claim boundaries Local evidence and status visualization	Separates declared intent from evidence-backed implementation.
ANALYSIS.md		Makes outputs inspectable without upgrading simulation into a hardware claim.
streamlit_app.py		

local cases; it does not establish accelerator correctness, real-device behavior, production reliability, or benchmark completion.

No numerical performance result is promoted by this test run. Where scripts emit JSON or JSONL, those outputs remain raw or simulation-specific until a reviewed summary includes hardware, software, workload, warm-up, repetition, correctness threshold, Git revision, and limitations.

VII. CLAIM BOUNDARIES AND RISKS

The project records these unverified or excluded claims:

- The simulator does not allocate real model tensors or measure device memory

The main risk is confusing synthetic or modeled behavior with deployed-system behavior. Other risks include incomplete workloads, platform-dependent timing, missing failure injection, and interfaces not yet exercised across device boundaries. Performance claims require a reviewed record meeting the portfolio standard.

VIII. EVALUATION PLAN

Evaluation proceeds through correctness tests, deterministic reproduction with Git and environment metadata, repeated benchmarks reporting latency/throughput/memory/error metrics, and a named profiler capture tied to exact hardware and source revision. Success requires reference equivalence within a documented tolerance, preservation of safety and resource invariants, clear failures, and evidence reproducible from a clean environment.

IX. GOALS, MILESTONES, AND SUCCESS CRITERIA

The project goals are staged so that correctness precedes performance and integration. A goal is complete only when its proof artifact is committed or otherwise reviewable; prose or a simulated number alone is insufficient.

X. FUTURE WORK AND INTEGRATIONS

The five project-specific next steps are:

- 1) Request growth and partial-page accounting
- 2) Fragmentation and eviction policies
- 3) Tensor-backed cache comparison
- 4) Integrate vLLM or SGLang request traces.
- 5) Export fragmentation and eviction events to ARGUS and SENTINEL.

The early items complete declared evidence; the later items connect downstream portfolio consumers. Each integration should add tests and a reviewable artifact such as JSONL evidence, a report, profiler capture, deployment manifest, trace, or labeled data set.

XI. CONCLUSION

AEGIS KV-Cache Lab is an evidence-aware active prototype: its implemented behavior and tests are real, while unbuilt hardware, deployment, and performance goals remain labeled. Completing the five integrations in dependency order will advance it from a learning artifact toward a credible portfolio component.

REFERENCES

- [1] Aaron Singh, "AEGIS KV-Cache Lab PROJECT.yaml," local portfolio repository.
- [2] Aaron Singh, "AEGIS KV-Cache Lab: README, ANALYSIS, source, and test artifacts," local portfolio repository.
- [3] Aaron Singh, "AI Infrastructure Portfolio Benchmark Standard," local portfolio repository.

TABLE II
AUDITED TEST MATRIX

Test artifact and case	Behavior being checked	Observed result
tests/unit/test_kv_cache.py:test_allocation_rounds_tokens_up_to_pages	Allocation rounds tokens up to pages.	Pass (local)
tests/unit/test_kv_cache.py:test_release_makes_pages_reusable	Release makes pages reusable.	Pass (local)
tests/unit/test_kv_cache.py:test_exhaustion_rejects_without_corrupting_cache	Exhaustion rejects without corrupting cache.	Pass (local)
tests/unit/test_kv_cache.py:test_duplicate_request_is_rejected	Duplicate request is rejected.	Pass (local)
tests/unit/test_kv_cache.py:test_unknown_release_is_rejected	Unknown release is rejected.	Pass (local)

TABLE III
DECLARED STATUS VERSUS AUDITED EVIDENCE

Audit field	Finding	Evidence source
Declared status	Active Prototype	PROJECT.yaml
Evidence-backed status	Active local prototype; 5 tests passed	Source plus local unittest run
Accepted measured results	None recorded in measured_results	PROJECT.yaml
Mismatch / claim boundary	The simulator does not allocate real model tensors or measure device memory	PROJECT.yaml and ANALYSIS.md
Next proof required	Request growth and partial-page accounting; Fragmentation and eviction policies	Planned features

TABLE IV
PROJECT GOALS AND REQUIRED PROOF

ID	Goal	Completion evidence
G1	Request growth and partial-page accounting	Passing tests and a reviewed source artifact
G2	Fragmentation and eviction policies	Machine-readable result with reproduction metadata
G3	Tensor-backed cache comparison	Reference-equivalence or domain-correctness report
G4	Model fragmentation, growth, and eviction	Named profiler, deployment, or integration artifact
G5	Compare against a tensor-backed runtime cache	Dashboard/report link preserving provenance and limitations