Layered Cluster-Compute Architecture

A Cost-First Verified Cluster-Kernel Lifecycle for Reducing Repetitive LLM Inference

Not Another Cache: A Verified Cluster-Kernel Lifecycle for Routine AI Work

Arcman
International Institute of Arc Theory

Version v0.4 – Extreme Edge-State Guards Edition
June 1, 2026

Document type	Technical Whitepaper / Engineering Proposal
Version	v0.4 DOI edition
Status	Conceptual engineering framework with pilot protocol
Intended use	Offline replay, shadow-mode testing, cost estimation, pilot planning
Not intended as	Production deployment guide without site-specific validation
DOI	10.5281/zenodo.20483510

Abstract

This whitepaper proposes **Layered Cluster-Compute Architecture** (LCCA), a cost-first and verification-aware lifecycle for reducing repetitive large language model (LLM) inference. The core idea is not merely to cache prompts or route requests, but to identify high-frequency verified workloads, compile them into versioned and auditable **cluster-kernels**, route routine tasks to these kernels, reserve frontier models for novel or uncertain work, and verify outputs before acceptance.

LCCA begins as a sidecar rather than a replacement. It can be tested through offline replay, shadow mode, read-only assist, and limited canary rollout. Its primary evaluation metric is **cost per verified useful output**, followed by Wh per verified output, token/TOTEN per verified output, latency, kernel hit rate, frontier escalation rate, and pilot risk magnitude.

The architecture is intended for high-frequency, stable, bounded, and verifiable workloads. It does not claim to replace foundation models, frontier reasoning, retrieval systems, caching systems, or tool-calling frameworks. Instead, it organizes these components into a measurable lifecycle with cost accounting, verification gates, amortization formulas, fallback paths, kernel governance, asymmetric low-cost verification, fast-path gateway routing, fractional drift probing, PromptOps command cards, TOTEN accounting, and night-cycle distillation. Version v0.4 further adds two extreme edge-state guards: in-flight request coalescing to prevent burst new-load stampedes, and probabilistic jittered degradation to prevent hot-kernel eviction tsunamis.

Core thesis: routine cognition should be compiled; frontier cognition should be reserved.

Keywords: Layered Cluster-Compute Architecture; LCCA; cluster-kernel; PromptOps; TOTEN; verified useful output; LLM inference cost; semantic caching; model routing; prompt caching; verification layer; cost per verified output; Wh per verified output; kernel lifecycle; night-cycle distillation; safe rollout; pilot risk magnitude; AI infrastructure; AI cost optimization; singleflight; in-flight request coalescing; thundering herd; stale-while-revalidate; jittered degradation; kernel eviction tsunami.

Contents

1 Problem: Token Brute-Force Inference **3**

2 Existing Techniques and the Gap **3**

3 Core Contribution: Verified Cluster-Kernel Lifecycle **4**

4 Architecture Overview **4**
4.1 New-load measurement . . . 5
4.2 Cluster before compute . . . 5
4.3 Sparse routing . . . 5
4.4 Verification gate . . . 5
4.5 Asymmetric low-cost verifier policy . . . 6
4.6 Gateway latency budget and fast-path short-circuit . . . 6

5 What Is a Cluster-Kernel? **7**

6 Kernel Lifecycle and Staleness Control **7**
6.1 Fractional canary probing and kernel drift control . . . 8

7 Extreme Edge-State Guards **9**
7.1 In-flight request coalescing for burst new loads . . . 9
7.2 Probabilistic jittered degradation for hot-kernel eviction . . . 10
7.3 Stale-while-revalidate and stale-if-error policy . . . 11
7.4 Cost accounting for edge-state guards . . . 11

8 Cost-First Model with Maintenance Cost **12**

9 Break-even and Minimum Hit Rate **12**

10 Tokenized and TOTENized Cost Metrics **12**
10.1 Raw token accounting . . . 13
10.2 TOTEN: Total Token-Equivalent Number . . . 13
10.3 PromptOps overhead ratio . . . 13

11 Watt / Wh Model **14**

12 PromptOps Layer: AI-Executable Operation Loops **14**
12.1 Prompt card format . . . 14
12.2 PromptOps safety boundary . . . 15

13 Pilot Risk Magnitude Index **15**

14 Safe Testing Ladder **15**

15 Ablation Benchmark **16**

16 Suitable and Unsuitable Workloads **16**

17 Security, Privacy, and Data Boundary **16**

18 Implementation Starter Kit **17**
18.1 Three example kernels . . . 17

19 Simulation Curves **17**

20 Thirty-Day Pilot Plan **22**
20.1 Week 1: Baseline . . . 22
20.2 Week 2: Offline replay . . . 22
20.3 Week 3: Shadow mode . . . 22
20.4 Week 4: Canary . . . 22

21 System Impact **23**
21.1 Compute . . . 23
21.2 Cost . . . 23
21.3 Latency . . . 23
21.4 Peak load . . . 23
21.5 Storage . . . 23

22 Impact on LLMs **23**

23 Boundary of Claims **24**

24 Conclusion **24**

A PromptOps Command Cards **25**
A.1 Prompt Card 1: Workload Baseline Extractor . . . 25
A.2 Prompt Card 2: Kernel Candidate Miner . . . 25
A.3 Prompt Card 3: Cost and Break-even Calculator . . . 25
A.4 Prompt Card 4: Token / TOTEN Cost Analyzer . . . 26
A.5 Prompt Card 5: Kernel Specification Generator . . . 26
A.6 Prompt Card 6: Verifier Designer . . . 26
A.7 Prompt Card 7: Pilot Risk Magnitude Scorer . . . 27
A.8 Prompt Card 8: Ablation Benchmark Planner . . . 27
A.9 Prompt Card 9: Dashboard Config Generator . . . 27
A.10 Prompt Card 10: Night-Cycle Distiller . . . 28

B Arc-Geometric Interpretation Layer **28**

C References **28**

1 Problem: Token Brute-Force Inference

Current LLM applications often route both routine work and frontier work through the same expensive full-model inference path:

$$\text{Prompt} \rightarrow \text{Tokenization} \rightarrow \text{Full LLM inference} \rightarrow \text{Generated output.}$$

This creates structural waste. Many tasks are frequent, stable, bounded, and verifiable, yet they are repeatedly recomputed from the token level. Examples include metadata generation, standard document revision notes, repository descriptions, customer support answers, schema transformations, report templates, low-risk summarization, and routine tool workflows.

This pattern may be called:

$$\boxed{\text{Token Brute-Force Inference.}}$$

The issue is not that LLMs are unintelligent. The issue is that the workload is poorly stratified:

$$\boxed{\text{The problem is not insufficient intelligence, but inefficient workload stratification.}}$$

LCCA therefore starts from a simple engineering principle:

$$\boxed{\text{Compile the routine; reserve the frontier.}}$$

2 Existing Techniques and the Gap

LCCA does not pretend that caching, routing, retrieval, or tools are new. Prompt caching is already used to reduce latency and cost for repeated prompt prefixes. Prefix/KV caching is a known serving optimization. Semantic caching has been studied as a way to reuse semantically similar responses. Model routing frameworks such as RouteLLM already address cost-performance tradeoffs by routing easier requests to cheaper models.

The gap is lifecycle integration. Existing techniques often solve one part of the problem, but they do not necessarily provide a full path from workload detection to verified kernel compilation, amortization, versioning, fallback, night-cycle distillation, PromptOps execution, TOTEN accounting, and pilot risk control.

Existing technique	**What it solves**	**What remains open**
Prompt / prefix caching	Avoids recomputing repeated prefixes	Does not manage semantic reuse, verification, amortization, or kernel lifecycle
Semantic caching	Reuses responses for semantically similar requests	Can mismatch intent; often lacks verifier, versioning, fallback, and kernel governance
Model routing	Sends simpler tasks to cheaper models	Does not compile recurring workflows into verified reusable kernels
RAG	Retrieves external evidence	Does not itself define cost breakeven or kernel lifecycle
Prompt templates	Reuses output format	Usually not versioned, audited, verified, or amortized

Workflow agents	Execute multi-step tool flows	May lack cost-first routing, verified-output accounting, and kernel governance
Tool calling	Improves correctness for specific operations	Does not decide when repeated work should become a reusable kernel
LCCA	Organizes the above into a lifecycle	Requires workload analysis, verifier design, kernel governance, and staged rollout

LCCA is not a replacement for cache, RAG, routing, or tools; it is a lifecycle layer above them.

3 Core Contribution: Verified Cluster-Kernel Lifecycle

The contribution of LCCA is not "layered computation" in the abstract. Its contribution is the engineering lifecycle:

$$\text{detect} \rightarrow \text{cluster} \rightarrow \text{route} \rightarrow \text{verify} \rightarrow \text{compile} \rightarrow \text{register} \rightarrow \text{reuse} \rightarrow \text{monitor} \rightarrow \text{retire}.$$

The five key differentiators are:

Lifecycle | Verification | Amortization | Night Distillation | Fallback.

These elements turn ordinary layered tricks into an engineering operating model.

Version v0.3 added three production-edge patches:

Verifier must be asymmetric and cheap.

The gateway must not become the bottleneck.

No kernel is immune to drift probing.

Version v0.4 adds two extreme edge-state guards:

Do not let burst new-load herds stampede the frontier.

Do not let hot-kernel eviction become a compute tsunami.

4 Architecture Overview

The LCCA request path is:

$$\begin{aligned} q &\rightarrow \text{Load Meter} \rightarrow \text{Clusterizer} \rightarrow \text{Sparse Router} \\ &\rightarrow \text{Kernel / Tool / Small Model / Full LLM} \\ &\rightarrow \text{Verifier} \rightarrow \text{Writeback} \rightarrow \text{Re-cluster}. \end{aligned}$$

4.1 New-load measurement

Let q be a request, $E(q)$ its embedding, and $\mathcal{M} = \{C_k\}$ the set of known clusters in memory:

$$L_{\text{new}}(q) = 1 - \max_{C_k \in \mathcal{M}} \text{sim}\big(E(q), C_k\big).$$

If $L_{\text{new}}(q) < \epsilon$, the request is likely routine and should first try cache, kernel, tool, or small-model routes. If $L_{\text{new}}(q) \geq \epsilon$, the request may require frontier escalation.

Measure new load before spending frontier compute.

4.2 Cluster before compute

Tokens are input cursors, not intelligence units:

$$\tau_i = \text{token}_i, \qquad x_i = E(\tau_i) \in \mathbb{R}^d.$$

Given $X = \{x_i\}_{i=1}^N$, the clusterizer maps token vectors to fewer higher-level clusters:

$$\mathcal{C}(X) = \{C_k\}_{k=1}^K, \quad K \ll N.$$

This reduces token-level combinatorics into cluster-level operations.

4.3 Sparse routing

Let $\mathcal{R}$ be a set of possible routes. A route may be a cache hit, cluster-kernel, tool chain, small model, full model, frontier model, or human review:

$$\rho(q) = \arg\min_{r \in \mathcal{R}} [\lambda_E E_r + \lambda_T T_r + \lambda_R R_r - \lambda_Q Q_r].$$

where E_r is compute or energy cost, T_r is latency, R_r is risk, and Q_r is expected verified quality.

4.4 Verification gate

Generated output is not accepted merely because it is fluent:

$$y = \text{Generate}(q, \rho(q)), \qquad a = \mathcal{V}(y; \mathcal{E}, \mathcal{T}, \mathcal{S}).$$

where $\mathcal{E}$ is an evidence store, $\mathcal{T}$ includes tools, tests, calculators, and compilers, and $\mathcal{S}$ includes schemas, types, and formal constraints.

$$\text{Accept}(y) = \mathbf{1}[\mathcal{V}(y) \geq \theta_{\text{verify}}].$$

Probability proposes; verification disposes.

4.5 Asymmetric low-cost verifier policy

Verification must not recreate the same cost structure that LCCA tries to avoid. If a verifier routinely calls a frontier model, the system may save a full-model generation only to spend the savings on a full-model judge.

The verifier must be cheaper, narrower, and more deterministic than the generator it verifies.

Recommended verifier tiers are:

Tier	Verifier type	Typical use
0	JSON Schema, regex, required-field checks	metadata, forms, structured output
1	AST parser, compiler, unit test, calculator	code, formulas, numeric claims
2	database lookup, citation check, deterministic retrieval	factual or evidence-bound claims
3	local classifier or sub-1B discriminator	low-cost semantic validation
4	small LLM judge	ambiguous low-risk cases
5	frontier LLM judge	rare high-risk or disputed cases only

Frontier LLM-as-a-judge should be an exception path, not the default verifier.
Define the kernel-route verifier tax:

$$\mathrm{VerifierTax}_K = \frac{c_{V,K}}{(c_F + c_{V,F}) - c_K}.$$

A conservative pilot target is:

$$\boxed{\mathrm{VerifierTax}_K < 0.2.}$$

The TOTEN analogue is:

$$\Omega_V = \frac{TOTEN_{V,K}}{TOTEN_{\text{full avoided}}} < 0.2.$$

If verification consumes more than 20 percent of the avoided full-model cost or TOTEN, the verification design should be simplified, restricted to higher-risk routes, or moved to deterministic tools.

4.6 Gateway latency budget and fast-path short-circuit

The entry gate must not become the new bottleneck. LCCA should not compute embeddings, search a large memory, and run semantic routing for every request if a cheaper deterministic route is available.

The gate must be cheaper than the inference it tries to avoid.

Recommended gateway order:

1. **Rule fast path**: route by endpoint, API route, template ID, task label, schema ID, or other deterministic metadata, without embedding.

2. **ANN semantic path**: use approximate nearest-neighbor search such as HNSW, FAISS, or equivalent in-memory vector retrieval for non-obvious requests.
3. **Fallback path**: if semantic routing is uncertain or the gateway exceeds its latency budget, route to the original full-model or small-model path.

Let:

$$t_G = t_{\text{rule}} + t_{\text{embed}} + t_{\text{ANN}} + t_{\text{route}}.$$

Require:

$$\boxed{t_G < \min(\tau_G, \alpha t_F).}$$

Here t_F is the full-model latency or TTFT proxy, τ_G is the deployment-specific gateway SLO, and α is the allowed gate-overhead ratio. A practical initial value is $\alpha = 0.05$.

No expensive semantic routing before trying cheap deterministic routing.

5 What Is a Cluster-Kernel?

A cluster-kernel is not merely a prompt template:

A cluster-kernel is a versioned, callable, testable, auditable routine compiled from repeated verified traces.

A valid cluster-kernel must include:

Requirement	**Meaning**
Owner	Responsible team or person
Version	Update and rollback control
Input schema	Accepted input structure
Output schema	Expected output structure
Verification rule	Validity check
Hit threshold	Activation condition
Fallback path	Recovery if uncertain
Cost profile	Estimated cost
Token/TOTEN profile	Estimated token-equivalent cost
Expiration rule	Retirement or revalidation trigger
Audit log	Trace of decisions and outputs

No owner, no version, no verifier, no fallback – no kernel.

6 Kernel Lifecycle and Staleness Control

A kernel must move through an explicit lifecycle:

candidate → shadow → canary → active → deprecated → retired.

No kernel lives forever. Each kernel must define validation metadata, including last validated date, revalidation interval, expiration policy, fallback path, and deprecation rule.

Listing 1: Example cluster-kernel registry entry.

```
kernel_id: zenodo_metadata_v1
version: 1.0.0
owner: doc-engineering
status: active

input_schema: schemas/zenodo_input.json
output_schema: schemas/zenodo_output.json
hit_threshold: 0.87

verification:
  - doi_format_check
  - latex_safe_text_check
  - required_fields_check

fallback: full_llm_metadata_route
cost_estimate_usd: 0.003
toten_estimate: 1150

last_validated: 2026-05-30
revalidation_interval_days: 30
expiration_policy: expire_if_not_validated
deprecation_rule: replace_after_two_failed_revalidations
```

6.1 Fractional canary probing and kernel drift control

High hit rate does not exempt a kernel from revalidation. If a kernel becomes too successful, it may silently route around frontier models and continue producing stale outputs after external policies, schemas, evidence sources, or business rules change.

> High hit rate does not exempt a kernel from revalidation.

For active kernels, LCCA should allocate a small probe probability:

$$p_{\text{probe}} \in [0.001, 0.02].$$

A practical starting range is 0.5 percent to 1 percent:

$$p_{\text{probe}} = 0.005 \sim 0.01.$$

For a probed request, compare the kernel output with a probe route such as a frontier model, trusted tool chain, or evidence-bound verifier:

$$D_K(q) = d(y_K, y_{\text{probe}}).$$

If:

$$D_K(q) > \theta_D,$$

then trigger revalidation, shadow mode, deprecation, or retirement.

Probe triggers should also include rising user correction rate, increasing verifier failure rate, policy or schema updates, evidence-source version changes, expired revalidation interval, downstream incident, or manual owner flag.

> No kernel lives forever, and no kernel is immune to fractional probing.

7 Extreme Edge-State Guards

The preceding lifecycle controls ordinary production risk. Extreme traffic and eviction conditions require two additional safeguards. These guards do not replace the main LCCA path; they protect the frontier model from transient flood states.

> Frontier capacity must not be broken by either burst new-load stampedes or hot-kernel eviction tsunamis.

7.1 In-flight request coalescing for burst new loads

A black-swan event can cause many similar requests to arrive before night-cycle distillation has had time to compile a kernel. If every request is treated as high new-load and escalated to the frontier model, LCCA may create a frontier stampede.

LCCA should therefore use **in-flight request coalescing** for short-window high-similarity requests. Only one leader request is allowed to enter the frontier path; compatible follower requests wait for the leader result or take a bounded fallback path.

Define a grouping key:

$$g(q) = H(tenant, route\_id, schema, risk\_class, semantic\_bucket, personalization\_tag).$$

Two requests may be coalesced only if they are compatible and sufficiently similar:

$$\text{Coalesce}(q_i, q_j) = 1$$

when:

$$\text{compatible}(q_i, q_j) = 1$$

and:

$$\text{sim}(E(q_i), E(q_j)) > \theta_{\text{coal}}.$$

Compatibility should be checked before similarity. Compatibility should include tenant boundary, permission domain, task type, output schema, risk class, personalization state, and privacy constraints.

> Coalesce similarity only after compatibility.

If m compatible requests arrive within a coalescing window τ_{coal}, frontier calls may be reduced from m to 1:

$$\Delta c_{\text{herd}} \approx (m-1)(c_F + c_{V,F}).$$

The herd suppression ratio is:

$$HSR = 1 - \frac{N_{\text{frontier}}^{\text{actual}}}{N_{\text{frontier}}^{\text{naive}}}.$$

A system should also record the follower wait penalty:

$$W_{\text{coal}} = \text{avg}(t_{\text{follower wait}}).$$

The coalescing window should be deployment-specific. A typical interactive range is 100–500 ms, with a hard timeout and fallback to avoid user-visible stalls.

Listing 2: Illustrative singleflight-style in-flight coalescing.

```
def route_with_singleflight(q):
    key = compatible_group_key(q)

    if in_flight.exists(key):
        return in_flight.wait(key, timeout=COALESCE_TIMEOUT)

    with in_flight.leader(key):
        y = frontier_or_tool_route(q)
        in_flight.publish(key, y)
        return y
```

7.2 Probabilistic jittered degradation for hot-kernel eviction

A hot kernel may carry a large fraction of routine traffic. If the kernel is hard-evicted after a drift alarm, that traffic may instantly fall back to the frontier model and create a compute tsunami.

Kernel invalidation must be rate-limited.

The kernel state machine should therefore avoid direct ACTIVE-to-DEAD transitions:

ACTIVE → SOFT_STALE → PROBING → DEGRADED → REBUILDING → ACTIVE / RETIRED.

Let a kernel K carry traffic fraction h_K, and let total request rate be $Q(t)$. If fraction $p_K(t)$ leaks from the kernel route back to the frontier route, extra frontier pressure is:

$$Q_F^{\text{extra}}(t) = h_K Q(t) p_K(t).$$

Let $B_F(t)$ be available frontier capacity budget. The leakage fraction must obey:

$$\boxed{h_K Q(t) p_K(t) \le B_F(t).}$$

Thus:

$$\boxed{p_K(t) \le \frac{B_F(t)}{h_K Q(t)}.}$$

A practical degradation ladder may be:

$$p_K(t) : 1\% \to 5\% \to 20\% \to 50\% \to 100\%,$$

but each step must pass the frontier-capacity budget before activation.

Expiration should be jittered rather than synchronized:

$$T_{\text{expire}}^{(K)} = T_0 + \epsilon_K, \qquad \epsilon_K \sim U(-J, J).$$

Kernel expiration should be jittered, not synchronized.

7.3 Stale-while-revalidate and stale-if-error policy

For low-risk tasks, a soft-stale kernel may continue to serve traffic while background revalidation or rebuilding proceeds:

$$K_{\text{stale}} \rightarrow \text{serve low-risk traffic while revalidating.}$$

If the frontier route is overloaded or unavailable, low-risk tasks may temporarily use stale-if-error behavior with audit marking. High-risk tasks should fail closed or require human review.

Serve stale only for low-risk tasks; fail closed for high-risk tasks.

Listing 3: Illustrative soft-expiry and jittered degradation route.

```
def route_kernel_with_soft_expiry(q, kernel):
    if kernel.state == "ACTIVE":
        return kernel(q)

    if kernel.state in ["SOFT_STALE", "PROBING", "DEGRADED"]:
        p = rebuild_leakage_rate(kernel)

        if random() < p and frontier_capacity_available():
            return frontier_rebuild_route(q)

        if is_low_risk(q):
            return serve_stale_with_audit(kernel, q)

        return human_or_safe_fallback(q)

    if kernel.state == "RETIRED":
        return fallback_route(q)
```

7.4 Cost accounting for edge-state guards

Edge-state guards are not free. Their costs should be included in LCCA accounting.

For kernel-route dollar cost, define:

$$c_K^{\text{eff}} = c_K + c_{V,K} + p_{\text{probe}}(c_P + c_{V,P}) + p_{\text{rebuild}}(t)(c_F + c_{V,F}).$$

For token-equivalent accounting:

$$TOTEN_K^{\text{eff}} = TOTEN_K + TOTEN_{V,K} + p_{\text{probe}} TOTEN_P + p_{\text{rebuild}}(t) TOTEN_F.$$

For energy accounting:

$$e_K^{\text{eff}} = e_K + e_{V,K} + p_{\text{probe}}(e_P + e_{V,P}) + p_{\text{rebuild}}(t)(e_F + e_{V,F}).$$

These terms prevent coalescing, probing, and rebuilding from becoming hidden costs.

v0.3 saves money and energy; v0.4 adds guards so the system can survive bursts and hot-kernel retreat.

8 Cost-First Model with Maintenance Cost

Capital hears dollars first. LCCA should therefore evaluate cost before energy. Let N be number of requests, h kernel hit rate, s small-model/tool route rate, and r frontier escalation rate, with $h + s + r = 1$.

Old architecture:

$$\$_{\text{old}}(N) = N(c_F + c_{V,F}).$$

Define the effective kernel-route cost including fractional probe overhead and soft-rebuild leakage:

$$c_K^{\text{eff}} = c_K + c_{V,K} + p_{\text{probe}}(c_P + c_{V,P}) + p_{\text{rebuild}}(t)(c_F + c_{V,F}).$$

Here c_P is probe-route cost, $c_{V,P}$ is probe-route verification cost, and $p_{\text{rebuild}}(t)$ is the capacity-aware fraction of hot-kernel traffic intentionally leaked to the frontier route during soft degradation or rebuilding.

New architecture:

$$\$_{\text{new}}(N) = C_{\text{compile}} + C_{\text{maintain}} + N[c_R + hc_K^{\text{eff}} + s(c_S + c_{V,S}) + r(c_F + c_{V,F})].$$

where C_{compile} is one-time kernel compilation cost and C_{maintain} covers maintenance, revalidation, governance, and rollback cost.

The true metric is not raw output cost but verified output cost:

$$\$_{\text{old}}^{\text{verified}} = \frac{\$_{\text{old}}(N)}{Nq_{\text{old}}}, \qquad \$_{\text{new}}^{\text{verified}} = \frac{\$_{\text{new}}(N)}{Nq_{\text{new}}}.$$

Primary metric: cost per verified useful output.

9 Break-even and Minimum Hit Rate

For a simplified kernel-versus-full-model system with $s = 0$ and $r = 1 - h$, LCCA becomes cheaper than full-model recomputation when:

$$h > \frac{(C_{\text{compile}} + C_{\text{maintain}})/N + c_R}{(c_F + c_{V,F}) - c_K^{\text{eff}}}.$$

Define:

$$h_{\min} = \frac{(C_{\text{compile}} + C_{\text{maintain}})/N + c_R}{(c_F + c_{V,F}) - c_K^{\text{eff}}}.$$

If $h > h_{\min}$, LCCA begins saving money.

For a repeated task class, the break-even number of calls is:

$$N^* = \frac{C_{\text{compile}} + C_{\text{maintain}}}{(c_F + c_{V,F}) - c_K^{\text{eff}}}.$$

If $N > N^*$, continuing full-model recomputation is economically irrational.

10 Tokenized and TOTENized Cost Metrics

Dollar cost is the first business-facing metric, but token-equivalent workload should also be measured because LCCA introduces routing prompts, verifier prompts, tool instructions, fallback calls, and night-cycle distillation prompts. A system may reduce full-model calls while accidentally increasing total prompt overhead.

10.1 Raw token accounting

Let Tok_F be average full-model route tokens, Tok_K average kernel route tokens, Tok_R routing tokens, $Tok_{V,F}$ verification tokens after full-model output, and $Tok_{V,K}$ verification tokens after kernel output.

Old architecture:

$$Tok_{\text{old}} = N(Tok_F + Tok_{V,F}).$$

New architecture:

$$\begin{aligned} Tok_{\text{new}} = {} & Tok_{\text{compile}} + Tok_{\text{maintain}} \\ & + N[Tok_R + h(Tok_K + Tok_{V,K} + p_{\text{probe}} Tok_P + p_{\text{rebuild}}(t) Tok_F) \\ & \quad + s(Tok_S + Tok_{V,S}) + r(Tok_F + Tok_{V,F})]. \end{aligned}$$

Tokens per verified output:

$$Tok_{\text{verified}} = \frac{Tok_{\text{total}}}{Nq}.$$

10.2 TOTEN: Total Token-Equivalent Number

Define:

$$\boxed{TOTEN = \text{Total Token-Equivalent Number.}}$$

For route j:

$$TOTEN_j = w_{in} T_{in,j} + w_{out} T_{out,j} + w_{ctx} T_{ctx,j} + w_R T_{R,j} + w_V T_{V,j} + w_T T_{tool,j} + w_{fb} T_{fallback,j}.$$

If no production weights are available, all weights may initially be set to 1 for a first-order estimate. For a kernel route with fractional probing and soft-rebuild leakage, add $p_{\text{probe}} TOTEN_P + p_{\text{rebuild}}(t) TOTEN_F$ to the kernel route total. Then:

$$TOTEN_{\text{verified}} = \frac{TOTEN_{\text{total}}}{Nq}.$$

10.3 PromptOps overhead ratio

Prompt-based automation should not become a new token sink. Define:

$$\Omega_{\text{prompt}} = \frac{Tok_{\text{PromptOps}}}{Tok_{\text{full avoided}}}.$$

A conservative initial target is:

$$\boxed{\Omega_{\text{prompt}} < 0.2.}$$

That is, PromptOps overhead should not exceed 20 percent of the avoided full-model tokens during a pilot.

11 Watt / Wh Model

After dollars and token-equivalent workload, evaluate energy. Strictly:

$$\text{Watt} = \text{power}, \qquad Wh = \text{energy}.$$

For route j, with average power P_j watts and runtime t_j seconds:

$$e_j = \frac{P_j t_j}{3600}.$$

Old architecture:

$$Wh_{\text{old}}(N) = N(e_F + e_{V,F}).$$

New architecture:

$$\begin{aligned} Wh_{\text{new}}(N) = {} & Wh_{\text{compile}} + Wh_{\text{maintain}} \\ & + N[e_R + h(e_K + e_{V,K} + p_{\text{probe}}(e_P + e_{V,P}) + p_{\text{rebuild}}(t)(e_F + e_{V,F})) \\ & \quad + s(e_S + e_{V,S}) + r(e_F + e_{V,F})]. \end{aligned}$$

Energy per verified output:

$$Wh_{\text{verified}} = \frac{Wh_{\text{total}}}{Nq}.$$

Energy saving ratio:

$$S_{Wh} = 1 - \frac{Wh_{\text{new}}/(Nq_{\text{new}})}{Wh_{\text{old}}/(Nq_{\text{old}})}.$$

$ first, TOTEN next, Wh second, verified output always.

12 PromptOps Layer: AI-Executable Operation Loops

LCCA should not merely tell engineers what to do. It should provide structured prompt commands that allow an AI assistant, under human supervision, to perform repetitive setup, analysis, kernel drafting, cost calculation, risk scoring, verifier design, and reporting tasks.

Prompts are not prose; prompts are executable operation contracts.

A PromptOps loop may be written as:

$$P_n(I_n) \to A_n \to V_n \to B_n \to I_{n+1},$$

where P_n is an operation prompt, I_n is input material, A_n is AI-generated output such as a YAML config or analysis table, V_n is verification or human review, and B_n is writeback into the next step.

12.1 Prompt card format

Every prompt card should include:

- **Role**: what the AI should act as.
- **Inputs**: files, logs, configs, or tables.
- **Task**: operation to perform.
- **Output format**: JSON, YAML, Markdown table, CSV, or shell commands.
- **Verification**: how the output should be checked.
- **Guardrails**: what the AI must not do.
- **Next step**: which prompt or human action follows.

12.2 PromptOps safety boundary

AI may draft, calculate, simulate, and recommend. Production mutation requires explicit human approval. AI should not directly modify production databases, access unredacted sensitive logs, activate new kernels, or bypass fallback and audit controls.

13 Pilot Risk Magnitude Index

Use **Pilot Risk Magnitude**, not "risk geometry." All variables may be normalized to $[0, 1]$:

$$\mathcal{R}_{\text{pilot}} = \frac{B \cdot C \cdot (1 + M + A + P + D + K)}{(1 + V)(1 + F)(1 + O)}.$$

Symbol	Range	Meaning
B	$[0, 1]$	blast radius / traffic exposure
C	$[0, 1]$	business criticality
M	$[0, 1]$	mutation risk / state write risk
A	$[0, 1]$	autonomy level
P	$[0, 1]$	privacy sensitivity
D	$[0, 1]$	drift risk
K	$[0, 1]$	kernel uncertainty
V	$[0, 1]$	verification strength
F	$[0, 1]$	fallback strength
O	$[0, 1]$	observability

Exposure, criticality, autonomy, privacy, drift, and kernel uncertainty increase risk. Verification, fallback, and observability reduce risk.

14 Safe Testing Ladder

Stage	Mode	User impact	Write action	Risk
0	Offline replay	none	none	very low
1	Shadow mode	none	none	very low
2	Read-only assist	low	none	low
3	Canary 1%	small	limited	medium-low
4	Routine kernel route	controlled	low-risk only	medium
5	Human-approved write	medium	human approval	medium
6	Autonomous write	high	automatic	high

LCCA should start at Stage 0 or Stage 1. Advance only if:

$$\mathcal{R}_{\text{pilot}} < \theta_R, \qquad q_{\text{verified}} \geq \theta_Q, \qquad \text{fallback success rate} \geq \theta_F.$$

15 Ablation Benchmark

Ablation is mandatory. It proves whether LCCA adds value beyond ordinary caching and routing.

Group	Architecture	Purpose
A	Full LLM baseline	Old brute-force baseline
B	Prompt / prefix cache only	Exact or prefix reuse
C	Semantic cache + basic router	Common engineering shortcut
D	RAG + tool verification	Evidence/tool path
E	LCCA without night distillation	Tests kernel routing and verifier
F	Full LCCA with night distillation and kernel lifecycle	Tests complete lifecycle

Key metrics:

$$\$_{\text{verified}}, \quad Wh_{\text{verified}}, \quad TOTEN_{\text{verified}}, \quad p50/p95,$$
$$h_{\text{kernel}}, \quad r_{\text{frontier}}, \quad q_{\text{verified}}, \quad \text{fallback rate}, \quad C_{\text{maintain}}.$$

Minimum success condition:

$$\$^{F}_{\text{verified}} \leq 0.8 \cdot \$^{C}_{\text{verified}}.$$

Alternative success condition:

$$\$^{F}_{\text{verified}} \approx \$^{C}_{\text{verified}} \quad \text{and} \quad q^{F}_{\text{verified}} > q^{C}_{\text{verified}}.$$

If neither condition holds, then for this workload, simpler cache/router is sufficient.

16 Suitable and Unsuitable Workloads

Suitable first targets are high-frequency, stable, verifiable, bounded, and repetitive. Examples include metadata generation, document engineering, enterprise FAQ, internal knowledge Q&A, support ticket routing, standard code scaffolding, report templates, and citation checking.

Unsuitable early targets are low-frequency, novel, high-risk, hard to verify, or strongly context-dependent. Examples include medical diagnosis, legal final judgment, high-risk financial action, autonomous trading, frontier research, and original theory creation.

LCCA first wins in repetition, not in novelty.

17 Security, Privacy, and Data Boundary

LCCA must not turn cost optimization into a data-governance risk. Minimum rules:

- Historical logs used for replay must be sanitized.
- PII, secrets, credentials, and regulated data must not be written into kernel registries.
- Kernel outputs must preserve provenance where correctness depends on evidence.
- High-risk write actions require human approval.
- Critical tasks should fail closed; routine tasks may fallback.
- Audit logs must record route, kernel version, verifier result, and fallback path.

A production pilot should define data retention policy, access control, redaction rules, kernel registry permissions, and audit export format.

18 Implementation Starter Kit

A minimal starter kit should include Docker demo, proxy gateway, replay benchmark, cost calculator, PromptOps command cards, three example kernels, dashboard, and kill switch.

Listing 4: Suggested starter repository layout.

```
lcca-starter/
  docker-compose.yml
  proxy_gateway/
  replay_benchmark/
  cost_calculator/
  promptops_cards/
  kernels/
    zenodo_metadata_kernel/
    formula_safe_description_kernel/
    revision_notes_kernel/
  verifier/
  dashboard/
  configs/
  README.md
```

18.1 Three example kernels

Kernel 1: Zenodo Metadata Kernel. Input: title, author, DOI, abstract, keywords, resource type, language, license. Output: repository description, keyword list, short description, metadata warnings. Verification: DOI format check, required fields, formula-safe text.

Kernel 2: Formula-Safe Description Kernel. Purpose: convert LaTeX-heavy descriptions into repository-safe plain text. Output checks: no raw `\mathbb`, no raw `\operatorname`, no broken dollar signs, no duplicated rendered/source formulas.

Kernel 3: Revision Notes Kernel. Purpose: generate version notes from document diffs. Checks: version number present, DOI status present, claim changes listed, file references correct.

19 Simulation Curves

Main text should include nine charts. All charts are illustrative only. Replace all parameters with production values before decision-making.

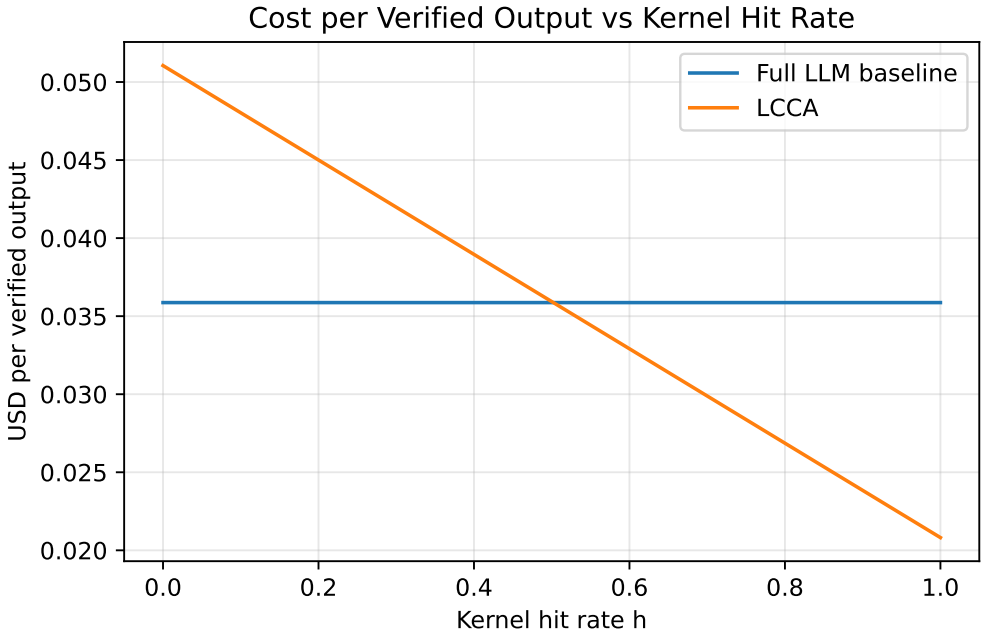

Figure 1: Cost per verified output vs kernel hit rate. Illustrative simulation only.

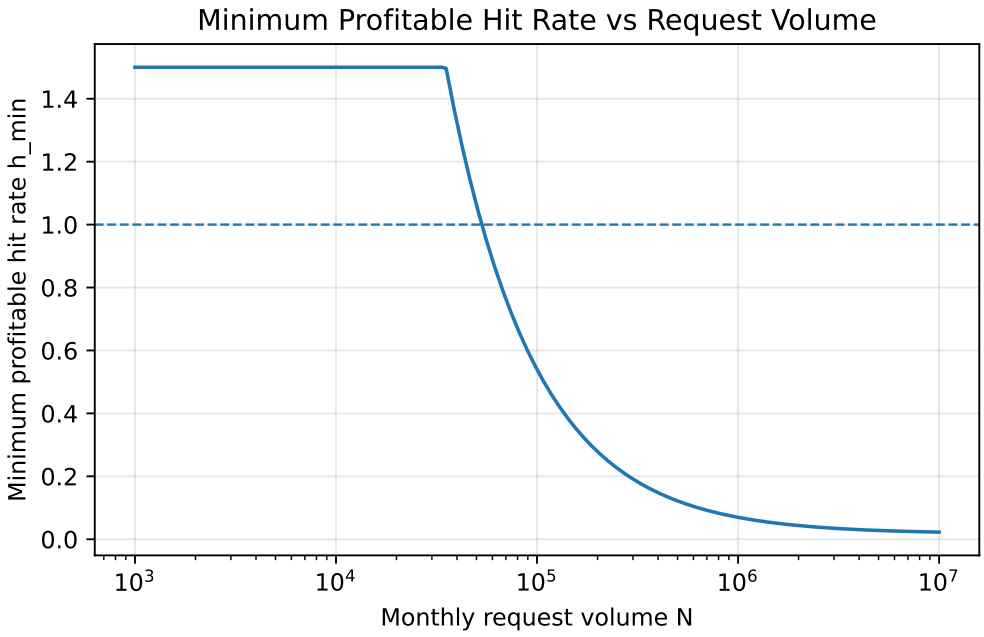

Figure 2: Minimum profitable hit rate $h_{\min}$ vs monthly request volume. Illustrative simulation only.

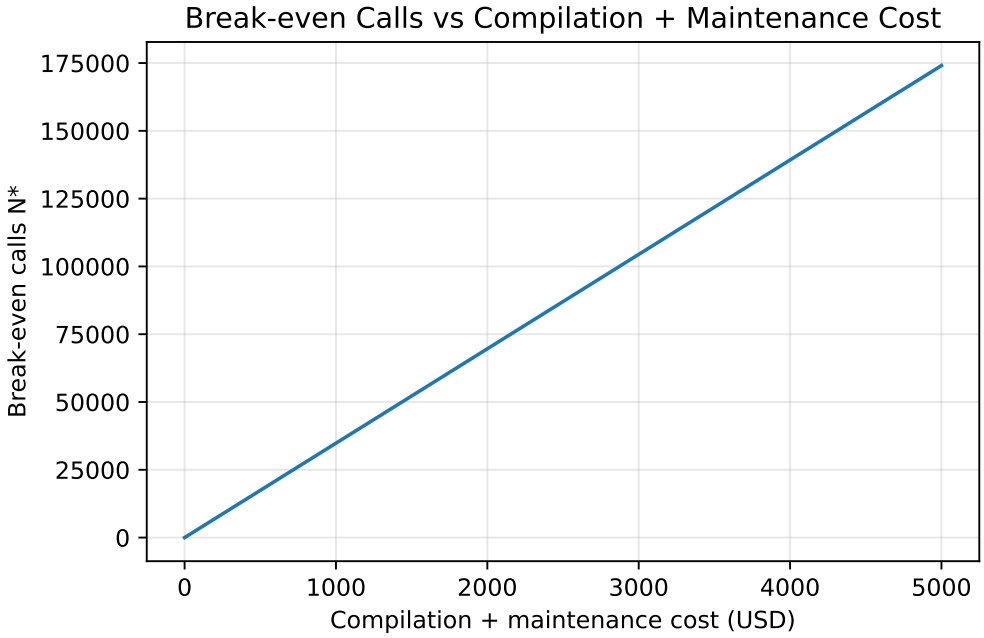

Figure 3: Break-even calls N^* vs compilation + maintenance cost. Illustrative simulation only.

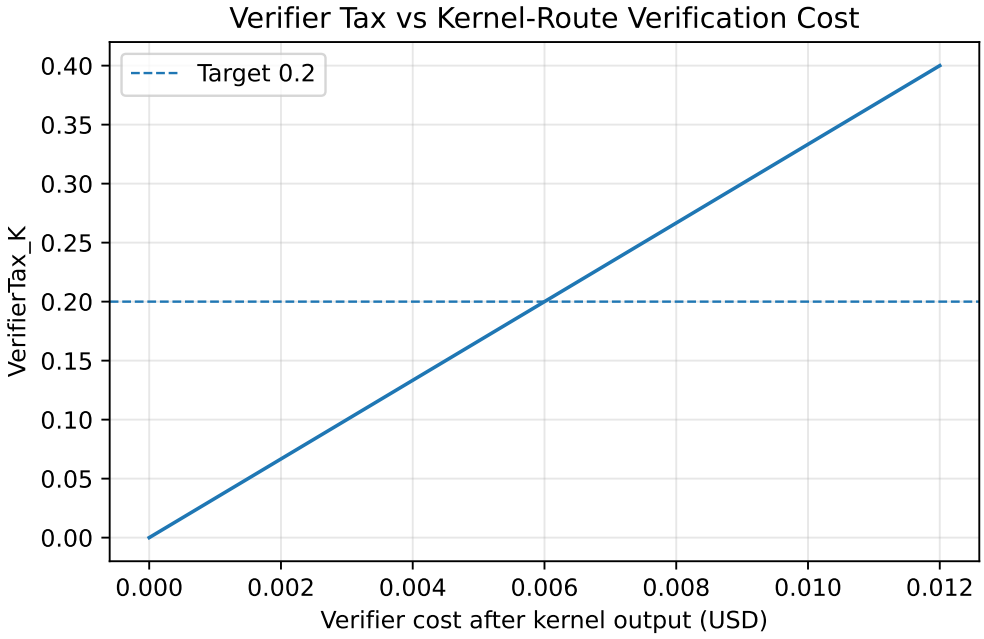

Figure 4: Verifier tax vs verifier cost share. Illustrative simulation only.

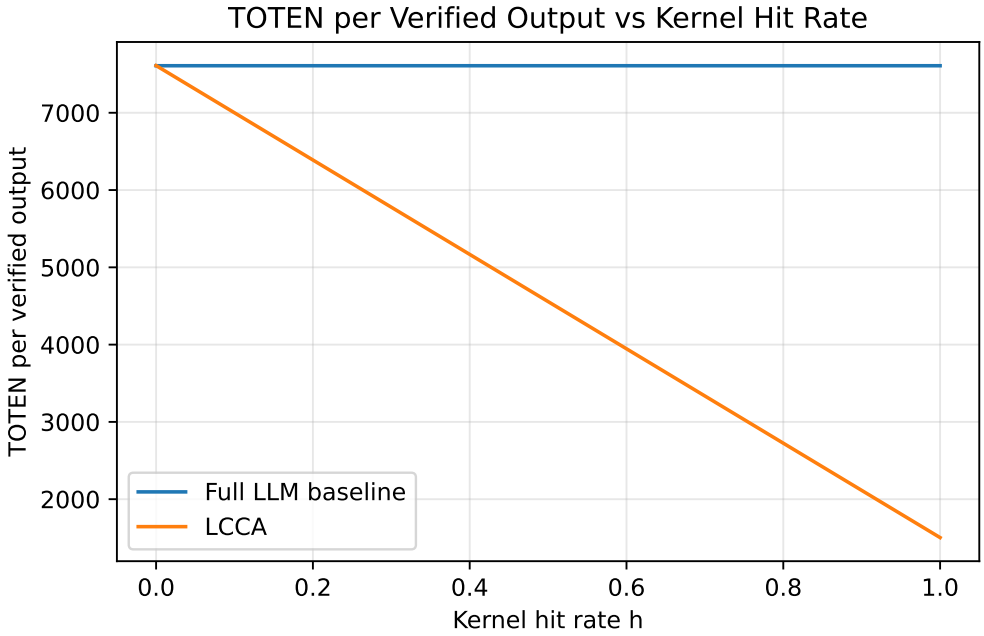

Figure 5: TOTEN per verified output vs kernel hit rate. Illustrative simulation only.

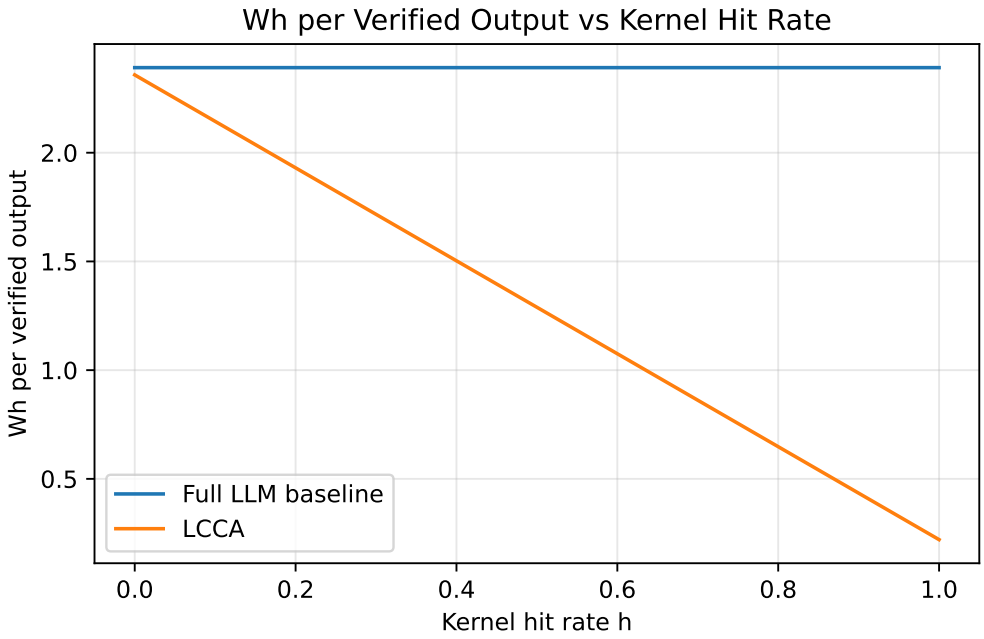

Figure 6: Wh per verified output vs kernel hit rate. Illustrative simulation only.

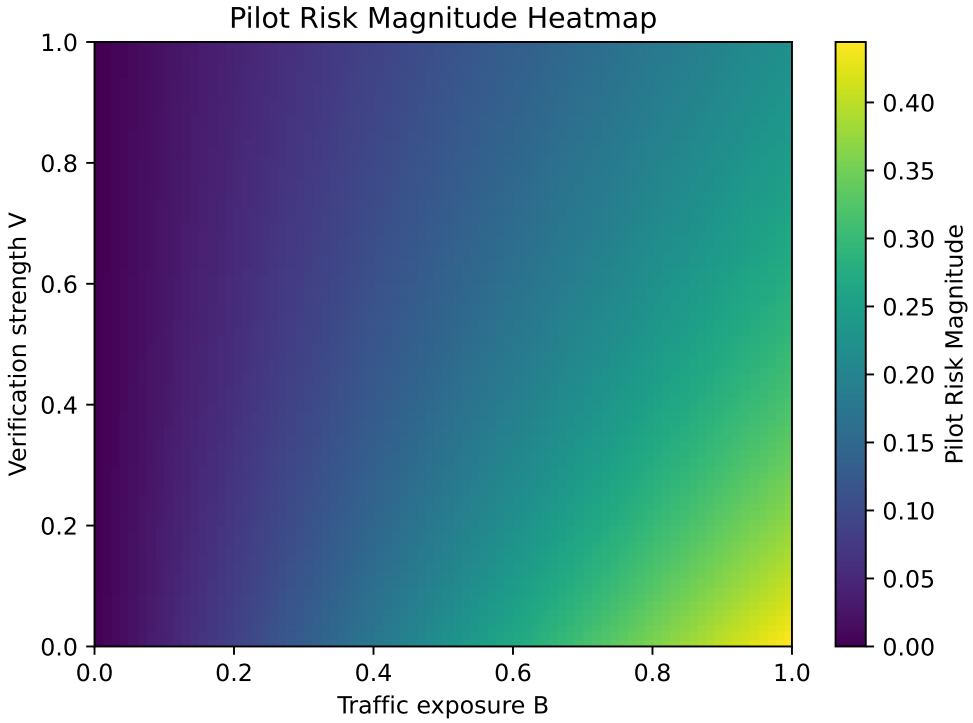

Figure 7: Pilot risk magnitude heatmap. Illustrative simulation only.

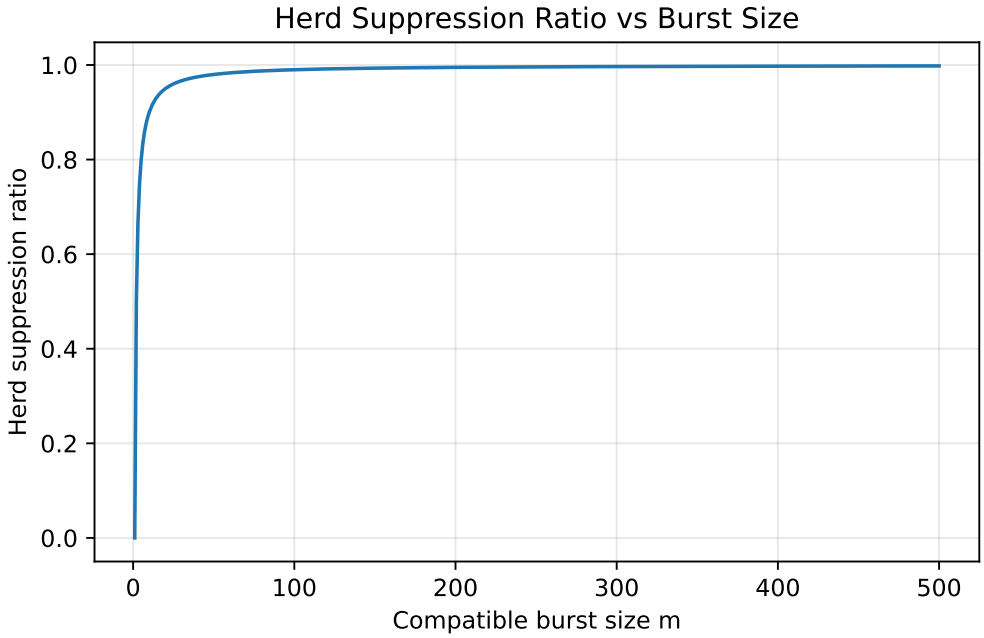

Figure 8: Herd suppression ratio vs burst size under in-flight request coalescing. Illustrative simulation only.

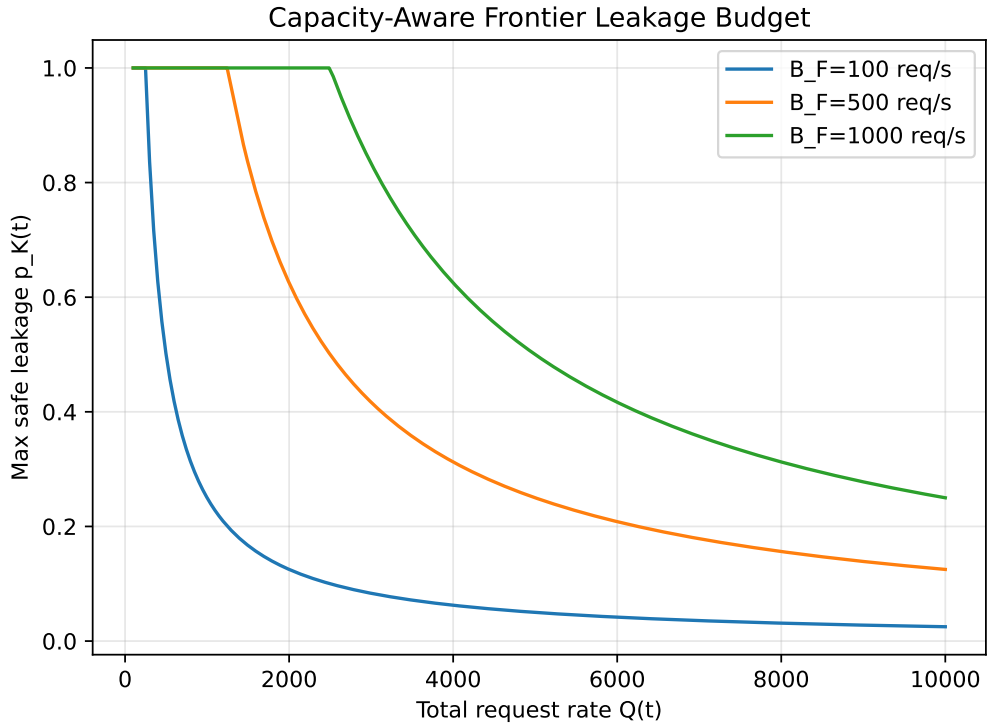

Figure 9: Capacity-aware frontier leakage budget during probabilistic jittered degradation. Illustrative simulation only.

20 Thirty-Day Pilot Plan

20.1 Week 1: Baseline

Collect N, c_F, t_F, Wh_F, q_{old}, task categories, full-model call distribution, and candidate repetitive task classes.

20.2 Week 2: Offline replay

Compute L_{new}, $h_{\text{potential}}$, N^*, $\mathcal{R}_{\text{pilot}}$, token/TOTEN estimates, and candidate kernel economics.

20.3 Week 3: Shadow mode

Run LCCA sidecar alongside production without affecting users. Track route recommendations, kernel hit simulation, verification pass rate, PromptOps overhead, and fallback simulation.

20.4 Week 4: Canary

Start with 1% low-risk traffic. Continue only if:

$$\$_{\text{verified}} \downarrow 20\%+, \qquad q_{\text{verified}} \geq q_{\text{old}}, \qquad \text{fallback success rate} \geq 99\%, \qquad \mathcal{R}_{\text{pilot}} < \theta_R.$$

21 System Impact

21.1 Compute

Expected in suitable repetitive workloads: $r_{\text{full}} \downarrow$, $h_{\text{kernel}} \uparrow$. A pilot target is to reduce full-model calls by 30%-70% in suitable repetitive workloads. This is a target range, not a universal guarantee.

21.2 Cost

The primary expected movement is $\$_{\text{verified}} \downarrow$.

21.3 Latency

Routine tasks may become faster:

$$t_{\text{routine,new}} \approx t_R + t_K + t_V.$$

Hard or risky tasks may become slower due to verification:

$$t_{\text{highrisk,new}} \approx t_F + t_V + t_{\text{tool}}.$$

Routine faster; uncertain safer.

21.4 Peak load

During peak load, the frontier threshold may increase:

$$\theta_{\text{frontier}}(t) = \theta_0 + \lambda P_{\text{load}}(t).$$

Then $P_{\text{load}} \uparrow \Rightarrow \theta_{\text{frontier}} \uparrow \Rightarrow r_{\text{full}} \downarrow$.

Version v0.4 adds two additional peak guards: coalescing high-similarity in-flight new-load requests, and degrading hot kernels probabilistically rather than evicting them all at once. Together, these prevent frontier capacity from being broken by either burst new-load stampedes or hot-kernel eviction tsunamis.

21.5 Storage

LCCA increases structured memory but reduces repeated inference. Storage layers include hot context, semantic cache, cluster memory, kernel registry, evidence store, verification logs, and raw logs with minimal necessary retention.

22 Impact on LLMs

LCCA does not remove LLMs. It repositions them.

Old role:

$$\text{LLM} = \text{default executor for every request.}$$

New role:

$$\text{LLM} = \text{frontier reasoner, kernel generator, complex synthesizer, exception handler.}$$

Routine full-model traffic may decrease, but high-value model use may increase: novel reasoning, kernel creation, complex synthesis, expert escalation, and frontier exploration.

The value unit may shift from generated tokens to verified useful outputs.

23 Boundary of Claims

LCCA is not a replacement for foundation models. It is not a claim that all reasoning can be cached, all tasks can become kernels, verification is free, or production deployment is safe without site-specific safeguards. It is a cost-aware, verification-aware, sidecar-first architecture for reducing repetitive LLM inference, with explicit guards for burst new-load herds and hot-kernel eviction tsunamis.

24 Conclusion

LCCA is not another cache. It is a cost-first verified cluster-kernel lifecycle for routine AI work.

Its central operational claim is:

$$\boxed{\text{Repeated verified work should not be repeatedly recomputed by frontier models.}}$$

Its economic claim is measurable:

$$\boxed{\text{Optimize cost per verified useful output.}}$$

Its engineering path is safe:

$$\begin{aligned}\text{offline replay} &\to \text{shadow mode} \to \text{read-only assist} \\ &\to \text{small canary} \to \text{controlled routine kernel routing.}\end{aligned}$$

Its strategic motto is:

$$\boxed{\text{Compile the routine; reserve the frontier.}}$$

Its extreme-edge operational guard is:

$$\boxed{\text{Coalesce burst new-load herds; degrade hot kernels with jitter.}}$$

A PromptOps Command Cards

The following cards are templates. They are not production commands. They should be adapted to local logs, security boundaries, and governance policies.

A.1 Prompt Card 1: Workload Baseline Extractor

```
Role:
You are an LCCA workload baseline analyst.

Inputs:
A JSONL or CSV file containing request_id, timestamp, prompt, route, model,
input_tokens, output_tokens, latency_ms, cost_usd, verification_status, and optional task_label.

Task:
1. Cluster requests into high-level task classes.
2. Estimate request count per task class.
3. Estimate average cost, latency, and token count per class.
4. Identify top 20 repetitive task classes.
5. Mark candidate classes suitable for cluster-kernel compilation.

Output Format:
Return JSON with task_classes, count, average_cost_usd, average_latency_ms,
average_input_tokens, average_output_tokens, estimated_repetition_score,
candidate_kernel_score, and notes.

Guardrails:
Redact PII. Do not expose sensitive user data. Do not recommend production deployment.
```

A.2 Prompt Card 2: Kernel Candidate Miner

```
Role:
You are an LCCA cluster-kernel mining agent.

Inputs:
Task class summaries, historical prompts, outputs, verification results, and cost data.

Task:
Identify candidate cluster-kernels from repeated verified task traces.
For each candidate, produce kernel_name, task_pattern, input_schema, output_schema,
estimated_monthly_calls, estimated_full_model_cost, estimated_kernel_cost,
required_verifier, fallback_route, estimated_break_even_N, suitability_score, and risk_notes.

Output Format:
YAML list of candidate kernels.

Guardrails:
Only recommend kernels for high-frequency, stable, bounded, verifiable tasks.
Do not recommend kernels for high-risk, novel, or hard-to-verify tasks.
```

A.3 Prompt Card 3: Cost and Break-even Calculator

```
Role:
You are an LCCA cost calculator.

Inputs:
N, c_F, c_K, c_R, c_VF, c_VK, C_compile, C_maintain, q_old, q_new.

Task:
Calculate old total cost, new total cost, cost per verified output, h_min,
```

```
N_star break-even calls, saving ratio, and recommendation.

Output Format:
Markdown table plus JSON summary.

Guardrails:
State clearly that values are estimates and must be replaced by production measurements.
```

A.4 Prompt Card 4: Token / TOTEN Cost Analyzer

```
Role:
You are an LCCA tokenization and TOTEN cost analyst.

Inputs:
For old and new routes, provide input_tokens, output_tokens, router_tokens,
verifier_tokens, tool_prompt_tokens, RAG_context_tokens, fallback_tokens,
compilation_tokens, maintenance_tokens, verified_success_rate.

Task:
Calculate total token count, token count per verified output, token avoidance ratio,
prompt overhead ratio, TOTEN score if weights are provided, and whether PromptOps
overhead cancels savings.

Output Format:
Markdown table and JSON.

Guardrails:
Distinguish raw token count from dollar cost and Wh energy. Do not claim token savings
if verification or routing prompts consume the avoided tokens.
```

A.5 Prompt Card 5: Kernel Specification Generator

```
Role:
You are an LCCA kernel specification engineer.

Inputs:
A kernel candidate description, sample inputs, sample verified outputs, and verifier requirements.

Task:
Generate a production-readable kernel specification with kernel_id, version, owner, status,
input_schema, output_schema, hit_threshold, verification_rules, fallback_route,
cost_estimate_usd, token_estimate, revalidation_interval_days, expiration_policy,
deprecation_rule, and audit_fields.

Output Format:
YAML.

Guardrails:
No kernel may be generated without owner, version, verifier, and fallback.
```

A.6 Prompt Card 6: Verifier Designer

```
Role:
You are an LCCA verifier designer.

Inputs:
Kernel specification, output schema, examples of correct and incorrect outputs.

Task:
Design verification rules including schema checks, required fields, deterministic checks,
external evidence checks, calculator/tool checks, confidence thresholds, failure actions,
```

```
and fallback policy.

Output Format:
YAML verifier configuration plus plain-English explanation.

Guardrails:
Verification must be stricter for write actions, regulated content, or high-risk outputs.
```

A.7 Prompt Card 7: Pilot Risk Magnitude Scorer

```
Role:
You are an LCCA pilot risk assessor.

Inputs:
For each proposed pilot task, provide normalized values in [0,1]:
B, C, M, A, P, D, K, V, F, O.

Task:
Calculate R = B*C*(1+M+A+P+D+K)/((1+V)*(1+F)*(1+O)).
Classify each task as safe for offline replay, shadow mode, read-only assist,
1% canary, requires human approval, or not suitable for pilot.

Output Format:
Markdown table and JSON.

Guardrails:
Do not recommend autonomous write for high-risk tasks.
```

A.8 Prompt Card 8: Ablation Benchmark Planner

```
Role:
You are an LCCA ablation benchmark planner.

Inputs:
System description, workload categories, baseline data, and available components.

Task:
Design an ablation benchmark comparing full LLM baseline, prompt/prefix cache only,
semantic cache + router, RAG + tool verification, LCCA without night distillation,
and full LCCA.

For each group, specify routing logic, metrics, sample size, failure modes, and success criteria.

Output Format:
Benchmark plan in Markdown plus CSV-style metric table.

Guardrails:
If full LCCA does not beat simpler cache/router on cost per verified output or verified success rate,
recommend not deploying full LCCA.
```

A.9 Prompt Card 9: Dashboard Config Generator

```
Role:
You are an LCCA dashboard configuration assistant.

Inputs:
Available logs, metrics, and output schemas.

Task:
Generate dashboard sections for cost per request, cost per verified output,
token/TOTEN per verified output, Wh per verified output, kernel hit rate,
```

```
frontier escalation rate, fallback rate, verification pass rate, p50/p95 latency,
and pilot risk magnitude.

Output Format:
Dashboard spec in YAML.

Guardrails:
Place dollar metrics before energy metrics. Show verified-output metrics before raw-output metrics.
```

A.10 Prompt Card 10: Night-Cycle Distiller

```
Role:
You are an LCCA night-cycle distillation agent.

Inputs:
Verified traces from the last cycle, kernel registry, failure logs, and task frequency statistics.

Task:
Deduplicate repeated traces, identify patterns suitable for new kernels, identify stale kernels requiring
revalidation, identify kernels to merge, deprecate, or retire, and generate next-day kernel update
    recommendations.

Output Format:
Night-cycle report with new kernel candidates, revalidation list, deprecation list,
expected savings, and risk notes.

Guardrails:
Do not activate new kernels automatically. Output recommendations only.
```

B Arc-Geometric Interpretation Layer

This appendix is optional for engineering readers. It is not required to implement LCCA.

Engineering term	Arc-geometric reading
Token	information cursor
Embedding	geometricization
Cluster	effective differentiated grouping
Router	path selection among active poles
Kernel	compiled cluster-instrument
Verifier	truth-accounting gate
Night cycle	zero-new-load return clustering
Frontier model	boundary-forging spear
TOTEN	total token-equivalent accounting of operational load
PromptOps	AI-executable operation loop

The engineering core and interpretive layer should remain separated. Practical readers can use the architecture without first accepting the broader interpretive framework.

C References

1. OpenAI. *Prompt caching – OpenAI API documentation.* Accessed May 31, 2026. https://developers.openai.com/api/docs/guides/prompt-caching

2. OpenAI. *Prompting – OpenAI API documentation.* Accessed May 31, 2026. https://developers.openai.com/api/docs/guides/prompting
3. vLLM Project. *Automatic Prefix Caching.* Accessed May 31, 2026. https://docs.vllm.ai/en/latest/design/prefix_caching/
4. Ong, I., Almahairi, A., Wu, V., Chiang, W.-L., Wu, T., Gonzalez, J. E., Kadous, M. W., & Stoica, I. *RouteLLM: Learning to Route LLMs with Preference Data.* arXiv:2406.18665, 2024. https://arxiv.org/abs/2406.18665
5. LMSYS. *RouteLLM: An Open-Source Framework for Cost-Effective LLM Routing.* 2024. https://www.lmsys.org/blog/2024-07-01-routellm/
6. Liu, X., Atalar, B., Dai, X., Zuo, J., Wang, S., Lui, J. C. S., Chen, W., & Joe-Wong, C. *Continuous Semantic Caching for Low-Cost LLM Serving.* Microsoft Research publication page. Accessed May 31, 2026. https://www.microsoft.com/en-us/research/publication/continuous-semantic-caching-for-low-cost-llm-serving/
7. Johnson, J., Douze, M., & Jegou, H. *Billion-scale similarity search with GPUs.* IEEE Transactions on Big Data, 2019. https://arxiv.org/abs/1702.08734
8. Malkov, Y. A., & Yashunin, D. A. *Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs.* IEEE TPAMI, 2020. https://arxiv.org/abs/1603.09320
9. Go Team. *singleflight package documentation.* Accessed June 1, 2026. https://pkg.go.dev/golang.org/x/sync/singleflight
10. Nottingham, M. *HTTP Cache-Control Extensions for Stale Content.* RFC 5861, 2010. https://www.rfc-editor.org/rfc/rfc5861