LLooMA: A Network-Native Orchestration Architecture for Low-Latency Distributed Language Intelligence

Version 1.1 - April 2026

Hassan Habib
BestBytes AI LLC
www.peerllm.com
hassan@bestbytes.ai

***Abstract*—LLooMA 1.0 (Low-Latency Orchestration of Models and Agents) is not a model artifact but an orchestration policy over a live network of community-operated hosts. Its central claim is that distributed language systems should be scaled along three axes rather than two: compute scaling, model scaling, and *intelligent scaling*. The first adds more machines, the second improves or expands the underlying models, and the third improves the execution policy that decides when to route directly, when to split a prompt into a dependency graph, when to race multiple hosts, when to compress context, and when to continue through fallback.**

LLooMA 1.0 is deliberately hybrid. Today its orchestration brain is centralized for determinism, governance, and latency control, while actual inference is decentralized across PeerLLM hosts. This paper formalizes that design as a network-native control plane for distributed language intelligence. It contributes four main ideas: (i) semantics-aware task decomposition into dependency-aware execution graphs, (ii) tail-latency reduction through race-based host execution under strict deadlines, (iii) a prefix-preserving continuation operator for seamless recovery across heterogeneous models, and (iv) an incentive-aware account of speculative execution in a decentralized inference economy. Because a public deployment trace is not yet available, we complement the architectural description with an analytical evaluation of the race policy under log-normal and Pareto latency models and specify the metrics that a real deployment should report.

***Index Terms*—Distributed AI, orchestration, decentralized inference, tail latency, task graphs, serving systems, PeerLLM, LLooMA**

I. Introduction

LLooMA[1] starts from a different premise than ordinary model papers. A conventional language model paper asks how to improve a parameterized artifact: larger context, better training data, stronger reasoning, lower serving cost. LLooMA asks a different question: what should the *system* do when language intelligence is realized across many independent machines that vary in latency, capability, uptime, and trust?

This shift matters because PeerLLM is not a single host serving a single model. It is a heterogeneous host network governed by one control plane and one economic layer. In such a setting, the dominant problem is not only inference quality. It is coordination. A useful system must decide whether a request should bypass orchestration, whether it should be decomposed into sub-tasks, which hosts should race, how long the race should be allowed to continue, how partial outputs should be preserved, and how distributed results should be made to appear singular to the user. That decision logic is where LLooMA lives.

The paper's core thesis is therefore simple: *LLooMA is an orchestration policy rather than a model artifact*, and *intelligent scaling* is a third systems axis beyond compute scaling and model scaling. Compute scaling adds hosts. Model scaling improves the models available to the network. Intelligent scaling improves the policy that converts one request into one execution strategy. In decentralized AI, this third axis is not optional; it is the difference between a pile of hosts and a language service.

LLooMA 1.0 is also intentionally hybrid rather than ideologically pure. Its orchestration brain currently runs centrally, while community hosts execute the underlying model calls in decentralized fashion [1], [2], [13]. This is not a compromise disguised as decentralization. It is the architecture. Centralization supplies low-latency coordination, safety mediation, and accounting; decentralization supplies elastic inference capacity and an open participation surface. Future versions may decentralize more of the control logic, but the present paper does not soften the hybrid design.

II. LLooMA 1.0 Execution Model

Operationally, LLooMA 1.0 implements a fourteen-phase lifecycle from ingress to payout accounting, but four mechanisms define most of its technical character: the split decision policy, host selection and racing, deterministic structural compression, and prefix-preserving fallback [13]. Where the public materials are underspecified, this section proposes one concrete design consistent with the published architecture and marks it as such.

A. Task-Splitting Decision Policy

LLooMA should not pay an orchestration tax on prompts that are already cheap to answer directly. The current architecture therefore begins with a fast-path bypass for very short prompts, single-sentence requests, or small output budgets [13]. A concrete policy consistent with that design is a two-stage gate. Stage 1 is deterministic: bypass if the request is short,

[1]The name *LLooMA* is unrelated to Meta's LLaMA/Llama model family; here it denotes *Low-Latency Orchestration of Models and Agents*.

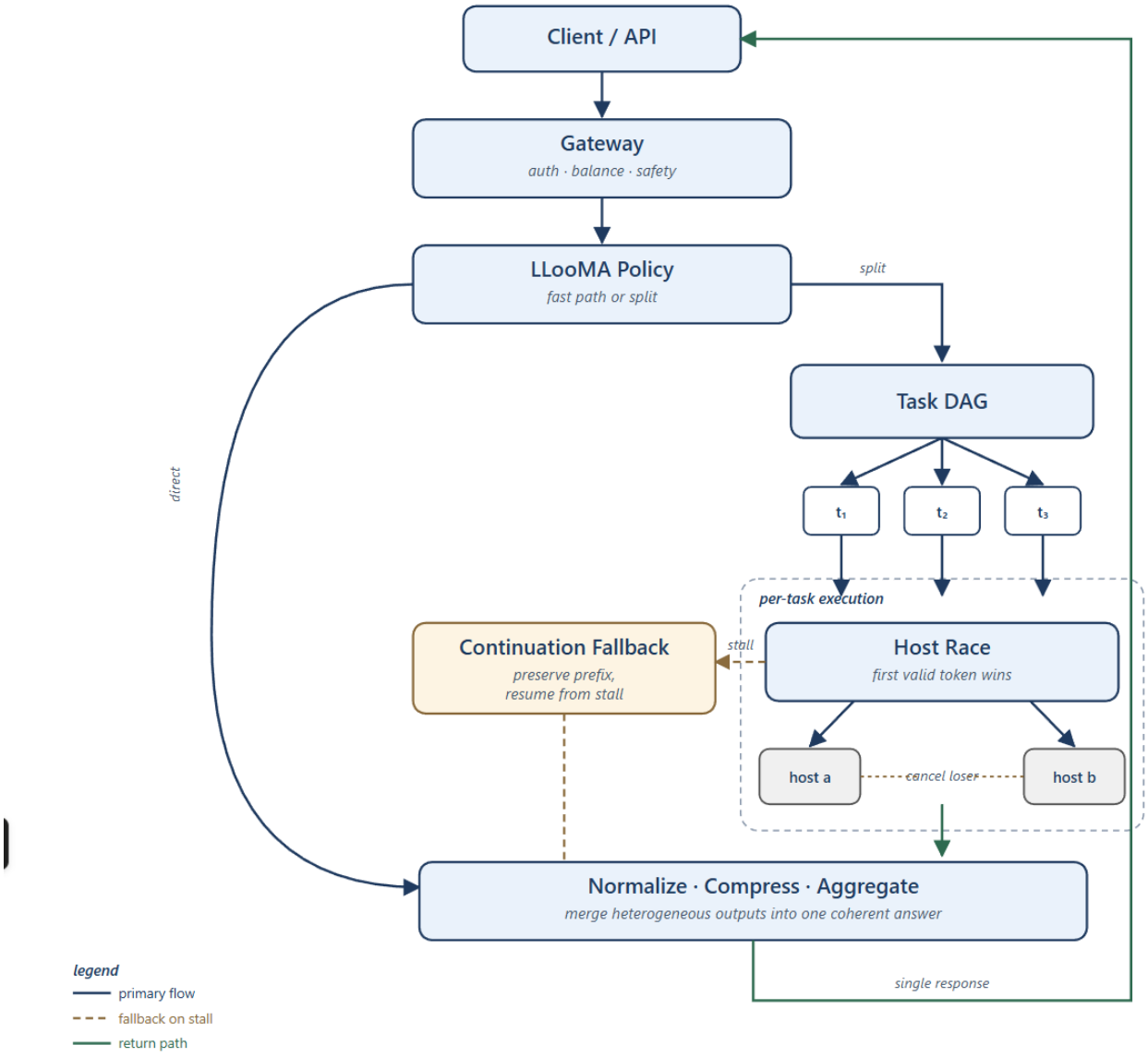

Fig. 1. LLooMA's execution path. A request either takes the direct fast path or is decomposed into a task DAG; each runnable node uses the same race primitive against eligible hosts; deadline or stall triggers continuation fallback; normalized outputs are finally aggregated into one response.

structurally simple, or capped at a small completion budget. Stage 2 is predictive: only requests that survive Stage 1 are scored for possible decomposition.

A practical scoring rule is to estimate whether splitting decreases critical-path latency enough to repay the split call itself. Let the splitter observe prompt length, sentence count, enumeration markers, multi-intent conjunctions, requested output format, and requested completion budget. It then estimates two quantities: a direct latency $\hat{L}_{dir}$ and a split latency $\hat{L}_{split} = t_{split} + t_{agg} + \max_P \sum_{v \in P} \hat{t}(v)$, where the maximum is taken over dependency paths in the candidate DAG. The orchestrator accepts decomposition only when $\hat{L}_{split} + \epsilon < \hat{L}_{dir}$ and the candidate graph satisfies cardinality and acyclicity constraints. Because the split call is executed only on requests already predicted to be structurally complex, its latency is amortized over the parallel work it unlocks rather than charged to the common case.

B. Host Selection and Race Dispatch

Each runnable task is dispatched to up to two eligible hosts. Eligibility is a hard predicate: approved model capability, fresh heartbeat, minimum software version, active connection, and no suspension [13]. Among eligible hosts, LLooMA needs a

concrete ranking rule. A reasonable design is to minimize a normalized host score

$$S(h) = 0.40\,\tilde{\rho}(h) + 0.30\,\widetilde{\ell}_{95}(h) + 0.20\,\tilde{s}(h) + 0.10\,\tilde{e}(h),$$

where $\tilde{\rho}$ is current in-flight load, $\widetilde{\ell}_{95}$ is recent first-token p95 latency, $\tilde{s}$ is heartbeat staleness, and $\tilde{e}$ is a timeout or error rate; all terms are scaled to $[0, 1]$. The orchestrator then samples two hosts without replacement from the best-scoring bucket to avoid deterministic concentration on a few nodes. The exact weights are a proposed instantiation rather than a public commitment.

C. Deterministic Structural Compression

Dependency injection and aggregation are only viable if intermediate outputs do not grow without bound. LLooMA's architecture describes structural compression without an additional AI call [13]. A concrete deterministic algorithm is as follows: (i) strip known control tokens and normalize whitespace; (ii) detect whether the text is prose, markdown, list-like, or JSON-like; (iii) preserve the structural skeleton, including headings, bullet prefixes, and JSON keys; (iv) deduplicate repeated lines or repeated fenced blocks; and (v) apply boundary-aware truncation that keeps roughly 60% of the budget from the prefix and 40% from the suffix, always cutting on sentence, list-item, or delimiter boundaries and inserting a fixed elision marker when content is removed. This is not summarization. It is deterministic shape-preserving compression whose advantage is predictable latency and zero extra model cost.

D. Continuation Operator Across Heterogeneous Models

LLooMA's most distinctive recovery mechanism is not generic fallback but *continuation*. When a host emits a useful prefix and then misses an inter-token deadline, LLooMA does not discard the work; it hands the prefix to a fallback model and asks it to resume from the exact interruption point. The phrase "exact interruption point" must be interpreted carefully in a heterogeneous network. Internal decoder state, sampler state, and token IDs are generally not transferable across different models or tokenizers.

A precise cross-model handoff therefore operates at the text surface rather than at hidden state. Let $y_{1:t}$ be the emitted byte stream when the host stalls. LLooMA first canonicalizes it to UTF-8, drops any incomplete trailing code point, and then trims back to the last safe boundary compatible with the current format: whitespace or sentence boundary for prose, delimiter boundary for JSON, and line boundary for markdown. The fallback request contains the original task, the preserved prefix, and a short tail window from that prefix. The fallback model is instructed to continue *after* the preserved prefix, maintain the same format, and avoid repeating the tail window. A rolling overlap filter on the generated suffix removes any repeated n-gram that duplicates the preserved tail. Tokenizer mismatch is thus resolved by re-tokenizing the same normalized text under the fallback model; hidden-state continuity is not claimed.

E. Privacy Mediation Threat Model

The public architecture already inserts centralized safety and privacy mediation before decentralized execution [13]. A concrete threat model makes that choice more rigorous. The decentralized host is treated as honest-but-curious by default and possibly malicious in the stronger model: it can log prompts, outputs, timestamps, and user metadata visible at the orchestrator boundary, but it cannot break transport security or compromise the orchestrator itself. The privacy mediator's job is therefore to remove direct identifiers (names, emails, phone numbers, addresses, account numbers), redact high-risk quasi-identifiers when feasible, and optionally neutralize stylistic fingerprints such as dialect markers or idiolect. The mediator does *not* claim to defend against semantic self-identification, downstream memorization by the centralized mediator, or endpoint compromise on the client side. LLooMA is a mediated system, not a confidential-compute system.

III. Analytical Evaluation of the Race Policy

A public deployment trace is not yet available, so this section analyzes LLooMA's dual-host race as a tail-latency control mechanism under standard response-time models. Let T_i denote the first-token latency of the ith eligible host for a given task and let $T_{(1)} = \min_i T_i$ be the observed race latency among k parallel hosts. Conditioning on the orchestrator's eligibility filter and score bucket, the order-statistics distribution is

$$F_{T_{(1)}}(t) = 1 - \left(1 - F_T(t)\right)^k. \tag{1}$$

If the orchestrator imposes a first-token deadline δ, the probability of centralized fallback is

$$\phi_k(\delta) = \Pr[T_{(1)} > \delta] = \left(1 - F_T(\delta)\right)^k. \tag{2}$$

Equations (1) and (2) already show the main effect: racing converts the tail of one host into the *kth power* of that tail.

A. Log-Normal Latency Model

For interactive inference, a log-normal first-token latency model is plausible because multiplicative factors such as queueing, scheduler delay, GPU contention, and prompt-length effects compound. If $T \sim \mathrm{LogNormal}(\mu, \sigma^2)$, the p-quantile of a k-host race is

$$q_p^{\mathrm{LN}}(k) = \exp\Big(\mu + \sigma\,\Phi^{-1}\Big(1 - (1-p)^{1/k}\Big)\Big), \tag{3}$$

where Φ^{-1} is the inverse standard normal CDF. Relative improvement depends only on k and the variance parameter σ:

$$R_p(k, \sigma) = \frac{q_p^{\mathrm{LN}}(k)}{q_p^{\mathrm{LN}}(1)}. \tag{4}$$

Table I instantiates this with a representative single-host median of 700 ms and first-token deadline $\delta = 1$ s. The dual-host race ($k = 2$) materially suppresses the tail even for moderate variance, while the gains become stronger as variance increases.

TABLE I

ANALYTICAL FIRST-TOKEN LATENCY FOR RACED HOSTS UNDER A LOG-NORMAL MODEL WITH SINGLE-HOST MEDIAN 700 MS AND DEADLINE $\delta = 1$ S. DUAL-HOST RACE IS THE DEPLOYED LLOOMA PRIMITIVE; $k = 3$ IS SHOWN ONLY TO ILLUSTRATE THE HOST-COUNT TREND.

σ	k	p50	p95	p99	$\phi_k(1\text{s})$
0.35	1	700	1245	1580	15.4%
0.35	2	578	913	1096	2.4%
0.35	3	525	787	922	0.4%
0.70	1	700	2214	3567	30.5%
0.70	2	478	1192	1717	9.3%
0.70	3	394	886	1215	2.8%

B. Heavy-Tail Robustness

Some hosts may exhibit genuinely heavy-tailed behavior due to thermal throttling, transient disk stalls, or background jobs. If $T \sim \text{Pareto}(x_m, \alpha)$ for $t \geq x_m$, then the p-quantile under a k-host race is

$$q_p^{\text{Par}}(k) = x_m(1-p)^{-1/(\alpha k)}, \tag{5}$$

and the expected race latency exists when $\alpha k > 1$ and equals

$$\mathbb{E}[T_{(1)}] = \frac{\alpha k\, x_m}{\alpha k - 1}. \tag{6}$$

For example, with $x_m = 250$ ms and $\alpha = 1.5$, the p99 first-token latency falls from about 5.39 s for a single host to about 1.16 s under a dual-host race. This is exactly the regime in which speculative duplication is attractive: the median is acceptable, but the tail is ruinous.

The qualitative conclusion is robust across both distributions. First, race benefits are dominated by variance reduction rather than mean reduction alone. Second, the value of racing rises as the tail grows heavier. Third, the first-token deadline interacts multiplicatively with host count through (2), so even modest per-host reliability improvements compound with racing. These results justify LLooMA's choice to spend a small amount of speculative compute in order to cut user-visible tail latency.

IV. ECONOMICS OF SPECULATIVE EXECUTION

The race policy creates an economic tension that a decentralized system cannot leave implicit. Public materials clearly account for winning-host contributions, but they do not fully specify whether a losing host in a race is compensated for speculative work [13]. The incentive consequences differ sharply.

Under a pure winner-take-all rule, the platform minimizes cost but pushes unrecovered speculative compute onto slower hosts. That rule is efficient only if losing hosts are still willing to participate despite frequently doing uncredited work. In equilibrium it risks adverse selection toward hosts that are already fast, near the orchestrator, or willing to tolerate low expected revenue. Slower but honest contributors may rationally leave the network or under-report availability.

At the opposite extreme, paying both raced hosts as if both had completed doubles the effective serving cost and largely nullifies the advantage of controlled redundancy. A more stable design is a two-part payment: a small reservation or cancellation credit for any host that accepts the race and begins work, plus a full completion payout for the winner. One concrete policy is a reservation fee plus a progress fee capped at a fraction η of the winner payout, with η initially set in the 0.10–0.20 range. This value is proposed here for analysis and should be replaced by observed network economics. Such a policy internalizes speculative cost without fully duplicating payout, keeps the platform's race premium bounded, and aligns host incentives with truthful participation.

TABLE II

MINIMUM DEPLOYMENT METRICS THAT A REAL LLOOMA EVALUATION SHOULD REPORT.

Metric	Definition and why it matters
p50/p95/p99 first-token latency	User-perceived responsiveness; must be broken down by direct path, split path, and fallback path.
Fallback rate	Fraction of requests or tasks that miss the deadline and invoke centralized completion. Measures network health and routing quality.
DAG split rate	Fraction of LLooMA requests that are decomposed. Needed to quantify how often intelligent scaling is actually exercised.
Orchestration overhead	Extra latency introduced by split analysis, DAG validation, dispatch, normalization, and aggregation relative to direct execution.
Race-win distribution	Share of wins by host, model family, geography, and score bucket; reveals concentration and possible routing unfairness.
Cost per successful response	User-side token cost plus host payout cost plus centralized fallback cost, normalized by completed responses.

The right metric is therefore not simply token payout per winner but *cost per successful response*, as listed in Table II. If racing reduces fallback and sharply reduces tail latency, a modest loser credit may still lower total cost by preventing expensive centralized completion and user abandonment.

V. RELATED WORK

LLooMA intersects several literatures but is not reducible to any one of them.

LLM serving systems. Systems such as Orca, vLLM, and SGLang improve throughput, batching, memory efficiency, and structured execution for model serving [3]–[5]. Their unit of optimization is typically the serving runtime for one model endpoint or one tightly managed cluster. LLooMA instead treats serving as a *network orchestration* problem over independent community hosts. Its main concern is cross-host policy under volatility, not only intra-engine efficiency.

LLM routing and cascade systems. FrugalGPT and RouteLLM learn when to send a query to a cheaper or stronger model in order to optimize cost-quality trade-offs [6], [7]. LLooMA shares the idea that requests should not all follow one execution path, but its routing unit is richer. It may split one request into multiple dependency-aware tasks, race multiple hosts on each task, and continue from partial decentralized output under hard deadlines. That is broader than single-shot model selection or cascade routing.

Tail-tolerant distributed systems. The dual-host race is directly informed by tail-latency work such as Dean and Barroso's discussion of hedged and tied requests, Vulimiri et al.'s analysis of redundancy for latency reduction, and Mantri's outlier-aware scheduling in MapReduce clusters [1], [2], [8]. LLooMA imports that logic into language inference, but with an important twist: the raced workers are heterogeneous models on community hosts, and the system must merge or continue text rather than merely return one idempotent RPC result.

Agent orchestration frameworks. LangGraph provides a graph-based runtime for long-lived, stateful agent workflows, and DSPy treats LM pipelines as declarative programs that can be optimized and compiled [9], [10]. These systems are closest in spirit to LLooMA's task-graph layer. The difference is deployment substrate and objective. LangGraph and DSPy primarily orchestrate tool- and model-calling workflows within one trusted application boundary. LLooMA uses graph execution as an online control policy over an unreliable decentralized inference market with explicit latency budgets, race execution, and host accounting.

Decentralized inference. Petals demonstrates collaborative inference by distributing layers of a single large model across many peers [11]. LLooMA is decentralized in a different sense. It does not shard one model across peers. It coordinates whole-task inference across many host-resident models and may combine their outputs through normalization, aggregation, and continuation. Petals is decentralized *model execution*; LLooMA is decentralized *execution policy over models and agents*.

The novelty claim can therefore be made precisely: to our knowledge, LLooMA is the first architecture in this space to combine *semantics-aware task splitting, dependency-aware decentralized execution, race-based tail suppression, prefix-preserving fallback across heterogeneous models, and payout-aware host accounting within one language-serving control plane*. None of the serving systems, routers, agent frameworks, or decentralized inference systems above supply that full combination.

VI. Conclusion

LLooMA is best understood as a policy layer that converts one language request into one execution strategy over a hybrid infrastructure. That framing keeps the paper's strongest idea at the center: intelligence in a distributed AI network is not only a matter of model weights or aggregate compute, but also of how the control plane interprets structure, allocates work, suppresses latency tails, preserves partial progress, and prices speculative effort.

This is why intelligent scaling deserves to be treated as a third systems axis. It explains when to avoid orchestration, when to decompose, when to race, and when to recover through continuation. In a network-native language system, those decisions are not implementation details. They are the system.

Acknowledgment

The author thanks the broader open-source AI community and the PeerLLM contributors whose practical constraints motivated LLooMA's hybrid design.

References

[1] J. Dean and L. A. Barroso, "The Tail at Scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[2] A. Vulimiri, O. Michel, P. B. Godfrey, and S. Shenker, "More Is Less: Reducing Latency via Redundancy," in *Proc. ACM HotNets*, 2012.

[3] G.-I. Yu, J. S. Jeong, G. Yoon, and B. Chun, "Orca: A Distributed Serving System for Transformer-Based Generative Models," in *Proc. USENIX OSDI*, 2022.

[4] W. Kwon *et al.*, "Efficient Memory Management for Large Language Model Serving with PagedAttention," in *Proc. ACM SOSP*, 2023.

[5] L. Zheng *et al.*, "SGLang: Efficient Execution of Structured Language Model Programs," arXiv:2312.07104, 2023.

[6] L. Chen *et al.*, "FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance," arXiv:2305.05176, 2023.

[7] I. Ong *et al.*, "RouteLLM: Learning to Route LLMs with Preference Data," arXiv:2406.18665, 2024.

[8] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Reining in the Outliers in Map-Reduce Clusters Using Mantri," in *Proc. USENIX OSDI*, 2010.

[9] LangChain, "LangGraph Overview," documentation, 2026. [Online]. Available: https://docs.langchain.com/oss/python/langgraph/overview

[10] O. Khattab *et al.*, "DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines," arXiv:2310.03714, 2023.

[11] A. Borzunov *et al.*, "Petals: Collaborative Inference and Fine-Tuning of Large Models," in *Proc. ACL System Demonstrations*, 2023.

[12] H. Habib, "PeerLLM: A Manifesto for a Decentralized AI Economy," PeerLLM Technical Report PLLM-TR-2025-01, Aug. 2025.

[13] PeerLLM, "LLooMA 1.0: Architecture and Request Lifecycle," PeerLLM Technical Report PLLM-TR-2026-01, Apr. 2026.