HiLL PUBLISHING

Engineering Advances, 2026, 6(4), 254-258
https://www.hillpublisher.com/journals/ea/
ISSN Online: 2768-7961

LLM-Assisted System Orchestration and Reliability Optimization Pathways in Complex Distributed Environments

Yunshu Zhang

Kit Circle, Austin, TX 78758, USA.

How to cite this paper: Yunshu Zhang. (2026) LLM-Assisted System Orchestration and Reliability Optimization Pathways in Complex Distributed Environments. *Engineering Advances*, 6(4), 254-258.
DOI: 10.26855/ea.2026.12.005

Received: September 9, 2026
Accepted: October 7, 2026
Published: November 3, 2026

***Corresponding author:** Yunshu Zhang, Kit Circle, Austin, TX 78758, USA.

© 2026 by the author(s).
This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution-NonCommercial-NoDerivatives (CC BY-NC-ND) license, which permits non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited and is not modified or adapted.
https://creativecommons.org/licenses/by-nc-nd/4.0/

Abstract

In complex distributed environments, the number of service units continues to expand, and the dependencies, resource states, and operational events interact with each other. System orchestration is no longer just about task dispatching and sequential execution; it also involves semantic recognition, risk constraints, anomaly suppression, and recovery coordination. Traditional control methods based on dependency scripts, rule trees, and fixed thresholds are prone to judgment delays, disconnection in handling, and reliability fluctuations when dealing with frequent changes, heterogeneous alerts, and link-level disturbances. The embedding of large language models (LLMs) adds context understanding, dependency reasoning, constraint generation, and feedback absorption capabilities to the orchestration links, forming enhanced support in task parsing, sequential generation, fault isolation, and recovery decision-making. Based on the system collaboration requirements in complex distributed environments, three reliability optimization paths can be further refined: orchestration advancement control, dynamic constraint generation, and recovery strategy linkage. These can be transformed into engineering benefits through controlled execution, indicator evaluation, and scenario verification.

Keywords

Large language model (LLMs); system orchestration; complex distributed environment; reliability optimization; intelligent operations and maintenance

After the convergence of cloud-native deployment, microservice splitting, cross-region collaboration, and continuous release, the operational focus of distributed systems has shifted from single-point stability to link stability [1]. The truly challenging aspect often lies not in whether a service is abnormal, but in whether the orchestration system can promptly identify the impact scope and contain the risk spread when the change actions, dependency relationships, and resource fluctuations are interrelated. The control methods that have long relied on manual experience patches and static rule concatenation have become increasingly sluggish in complex scenarios, especially when dealing with unstructured alerts, cross-component disturbances, and continuous gray-scale transitions. The response chains often suffer from understanding gaps and response mismatches. Therefore, introducing an auxiliary layer with semantic integration and context inference capabilities has become a practical direction to improve system collaboration quality and operational resilience.

1. LLM-Assisted Orchestration Mechanism in Complex Distributed Environments

1.1 Semantic Parsing and Intent Mapping for Heterogeneous Tasks

The orchestration entry in complex distributed environments often includes release single requests, alarm texts,

rollback instructions, configuration change records, and duty instructions. The expression granularity of the same event in different systems is not consistent. Relying solely on keyword matching can easily misjudge "interface timeout spread" as "single service load abnormality". After LLM (large language model) intervention, it first extracts slots for service names, versions, affected interfaces, dependent objects, execution windows, and constraints, and then compresses the natural language description into a standard task card, such as "payment-service—v2.3.7—gray release—associated 6 downstream interfaces—limitation threshold 15%—observation window 10 min", establishing a unified semantic foundation for subsequent orchestration, reducing path deviations caused by input ambiguity [2].

1.2 Orchestration Reasoning and Sequential Generation for Dependency Links

After the task is identified, it is not the number of actions that truly determines the quality of orchestration, but whether the sequence aligns with the dependency relationships and resource states. In this stage, the LLM needs to interact with service topology, configuration references, database connection pool levels, message accumulation depth, and historical change samples to split the execution chain, first confirming the necessary serial pre-check items, then identifying the parallelizable low-coupling steps, and adding fallback branches for high-risk nodes. For example, in the case of cross-service version switching, if the upstream interface field compatibility rate is lower than 98% or the cache synchronization delay exceeds 200 ms, the execution sequence should not directly enter the scaling stage but should remain in the verification branch, first holding back the chain offset.

1.3 Multi-Agent Collaborative Control Chain for Execution Loop

To reduce decision overlap under high-frequency changes, the execution loop is decomposed into four coordinated roles: a parsing agent, a strategy agent, a verification agent, and an exception agent. Given a task card T and runtime state S, the parsing agent extracts intent and dependency constraints; the strategy agent maps them into a release plan $P=\{r,b,o\}$, where r denotes gray-scale ratio, b denotes batch sequence, and o denotes observation window, and the verification agent evaluates P against configuration synchronization rate, field compatibility, and health-check status. If verification passes, P enters controls execution; otherwise, the exception agent generates fallback actions such as rate limiting, rollback, or traffic redirection. Manual confirmation, rule review, and audit logging remain mandatory at the execution boundary to prevent LLM outputs from bypassing production control constraints [3].

2. Reliability Optimization Path in Complex Distributed Environments

2.1 Risk Convergence Under Pre-Arranged Control

In complex distributed environments, reliability instability often originates from insufficient identification of version coupling, resource reserves, and dependency vulnerabilities at the orchestration entry stage [4]. To formalize pre-execution risk convergence, a composite score can be defined as $R_c=w_1(1-C_f)+w_2(1-R_r)+w_3F_{\mathrm{h}}+w_4D_v$, where C_f denotes field compatibility,R_r denotes resource reserve ratio, F_{h} denotes recent failure frequency, and D_v denotes dependency vulnerability. Before release, the LLM-assisted chain interprets interface compatibility matrices, node CPU levels, connection pool occupancy, and historical failure samples to estimate R_c. If R_c exceeds threshold θ, the orchestration flow is diverted to batch verification, warm-up probing, or dependency isolation rather than direct ramp-up, thereby shifting uncertainty resolution to the pre-execution stage and limiting the amplification of minor deviations along the call chain.

2.2 Fault Suppression Under Dynamic Constraint Generation

When reliability control remains limited to fixed thresholds and unified scripts, handling actions tend to lose precision under link jitter, traffic surges, or cross-region latency drift. A tighter formulation is to model constraint generation as $\Gamma=f(L,E,H,K)$, where L denotes latency drift, E denotes error escalation, H denotes health-check status, and K denotes service coupling level. The generated constraint set Γ determines canary ratio, retry cap, observation window, and cross-region switching permission at runtime. In practice, this may reduce the initial canary traffic to 5%, extend the health-check window to 8 min, cap retries at 2, and append configuration synchronization checks for highly coupled services. In this way, the orchestration engine receives bounded execution constraints rather than generic alarms, so fault diffusion is confined to a narrower local region [5].

2.3 Resilience Enhancement Under Recovery Strategy Linkage

After the abnormality enters the explicit stage, the recovery quality depends on whether the action sequence aligns with the fault domain boundary and business priority, rather than simply pursuing rollback speed [6]. As shown in Figure 1, the recovery chain assisted by LLM usually unfolds in the order of "root cause clue aggregation - fault domain segmentation - disposal strategy matching - recovery result injection": First, summarize log summaries, tracking spans, circuit breaker records, and index mutation points to identify the affected service clusters. Then, based on the transaction link weight, determine the order of throttling, degradation, flow cutting, or rollback. For example, the core payment link is prioritized for recovery, and the edge recommendation link is reconstructed later, compressing the average recovery time to a more controllable range, while reducing the secondary disturbances caused by incorrect rollbacks and repeated switching.

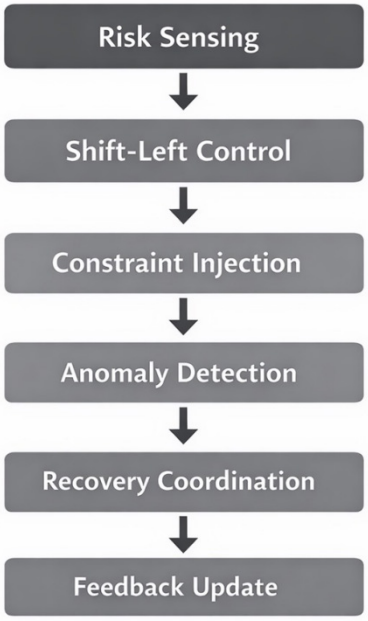

Figure 1. Closed-loop path for LLM-assisted reliability optimization.

3. LLM-Assisted Engineering Implementation and Performance Verification

3.1 Implementation Boundary Design Under the Controlled Execution Architecture

During the engineering implementation stage, the LLM should be placed before the orchestration engine and after the event collection layer, so that it only participates in task interpretation, constraint completion, and strategy candidate generation, without directly occupying the execution rights; during actual deployment, the actions can be divided into 4 levels: L1 is log aggregation and field completion, L2 is gray-scale ratio, batch sequence, and observation window suggestions, L3 is controlled script triggering, and L4 is database write path switching, cross-region traffic migration, and core configuration coverage, among which L1 and L2 allow automatic transition, L3 requires rule engine review before triggering, and L4 must be bound with double-person approval and rollback plan before being issued. To prevent the model output from exceeding the authority and entering the production environment, the execution chain also needs to embed threshold gates and audit numbers; for example, when the instance removal ratio exceeds 15%, the flow cut range exceeds 20% of the single-region traffic, and the health check fails for 3 consecutive times, the system automatically blocks the action push, and the candidate solution, review result, approval record, and execution hash are written to the audit library, with a retention period of no less than 180 days for traceability [7].

3.2 Reliability Evaluation Indicators for System Results

Reliability assessment should not remain at response quality, but should directly measure orchestration-chain

outcomes. In practice, indicators may be organized into four layers. The input layer records task-slot completion rate, dependency-identification recall, and constraint-extraction accuracy, with thresholds set at no less than 97%, 95%, and 94%, respectively. The orchestration layer tracks sequence-adoption rate, conflicting-step interception rate, and canary-rhythm deviation. The execution layer records release success rate, erroneous-operation interception rate, consistency-verification pass rate, and alert density per 1,000 orchestration actions. The recovery layer tracks MTTD, MTTR, repeated rollback rate, and secondary-disturbance incidence, using the most recent 30 similar changes as the baseline window and generating weekly comparison reports. To prevent single-metric fluctuation from masking overall drift, an overall reliability score can be defined as $S=0.2I+0.3O+0.3E+0.2R$, where I, O, E, and R denote the input, orchestration, execution, and recovery scores, respectively. Only when $S \geq 90$ for three consecutive evaluation periods should the LLM-assisted chain enter expanded deployment [8].

3.3 Construction of Verification Methods for Typical Scenarios

During verification, three high-frequency scenarios—gray-scale release, cross-service linkage change, and fault-recovery drill—should be used to construct a unified evaluation workflow. In the gray-scale release scenario, services with frequent changes in the past 90 days and dependency depth of no less than four layers are selected, and both the traditional rule chain and the LLM-assisted chain are executed for 20 rounds each. Metrics include initial canary ratio, health-check convergence time, abnormal diffusion nodes, and interruption frequency. Across the 20 runs, the LLM-assisted chain reduced abnormal diffusion nodes from 3.8 to 1.9 per release, shortened health-check convergence time from 11.4 min to 8.6 min, and lowered interruption frequency by 27.3%. In the cross-service linkage change scenario, dependency-identification deviation, conflicting-step interception rate, and configuration-synchronization success rate are measured under three injected disturbances—field missingness, cache delay, and message backlog. Each load test lasts 30 min at 1.2×–1.5× baseline traffic, under which dependency-identification deviation decreased from 7.1% to 3.2%, while configuration-synchronization success rate increased from 91.8% to 96.4%.

For the fault-recovery scenario, verification should address not only rollback completion but also the accuracy of recovery ordering and fault-domain segmentation [9]. A replay-based test set can be constructed from historical incidents, with fault portraits generated from log summaries, trace spans, circuit-breaker records, and metric mutation points. The LLM-assisted chain first produces a handling sequence, which is then blindly compared with the existing manual plan. Scoring items include localization time, rollback count, business impact, and secondary-disturbance rate, with a full score of 100; candidate plans scoring below 85 are excluded from production drills. After comparison, the SRE team reviews constraint omissions, sequence dislocations, and over-conservative decisions, and writes the corrections back into the sample library for retesting over the next two statistical periods. In the replay-based recovery set, median MTTR dropped from 18.7 min to 15.9 min, repeated rollback rate declined from 6.2% to 2.8%, and secondary-disturbance incidence fell by 21.5%. Only when MTTR reduction remains above 12%, and repeated rollback rate stays below 3% should the verification chain enter regular operation.

4. Conclusion

In a complex distributed environment, the key to system orchestration is no longer whether the task dispatch is completed, but whether the change interpretation, dependency identification, execution constraints, and recovery sequence can be stably connected within the same control chain. After embedding LLMs at the orchestration front end and the handling middle section, the depth of task understanding, risk convergence speed, and abnormal recovery sequence have obtained a more refined scheduling basis. However, its engineering value still depends on whether the permission layering, index verification, and scenario verification are synchronously implemented. In the future, it can continue to deepen along multi-modal observation fusion, online constraint correction, and cross-environment strategy migration, gradually moving the auxiliary orchestration from local enhancement to full-chain steady-state support.

References

[1] Sharma P, Sharma KS, Dani D. Edge-assisted federated learning for anomaly detection in diverse IoT network. Int J Inf Technol. 2024;17(5):3035-45

[2] Xu J, Luo J, Cao X, Gao Y, Mao S, Wang M, et al. Cooperative task scheduling and resource allocation of embodied multi-satellite systems: AI-driven perspective. Sci China Technol Sci. 2026;69(1):1100301.

[3] Jiao H, Gong J, Li H, Lam JSL, Shu Y, Wang J, et al. LLM4STP: A large language model-driven multi-feature fusion method for ship trajectory prediction. Transp Res Part E. 2026;207:104599.

[4] Bao Y, Du S, Jiang Q, Li Y, Zhao B, Wang G, et al. ESCAPE: an efficient and safe distributed UAV swarm exploration framework with collision avoidance perception. Auton Intell Syst. 2025;5(1):1-13.

[5] Sanyadanam A, Srirama NS. Comparison of distributed machine learning frameworks in a fog environment: conceptual and performance analysis. Internet Things. 2025;34:101774.

[6] Gupta M, Kumar M, Dhir R. Fedmed-xai: a collaborative and trustworthy framework for skin cancer detection using federated learning and explainable AI. J Supercomput. 2025;81(15):1437.

[7] Yadav A, Sharma A. Enhancing cloud orchestration using hybrid scheduling for QoS-aware and energy-efficient task allocation. Cluster Comput. 2025;28(13):859.

[8] Jin X, Teng J, Chen J. IoT-edge hybrid architecture with cross-modal transformer and federated manifold learning for safety-critical gesture control in adaptive mobility platforms. Future Internet. 2025;17(7):271.

[9] Zikai Z. Enhancing distributed machine learning through data shuffling: techniques, challenges, and implications. ITM Web Conf. 2025;73:03018.