International Journal of Advanced Engineering, Management and Science (IJAEMS)
Peer-Reviewed Journal
ISSN: 2454-1311 | Vol-12, Issue-2; Mar-Apr, 2026
Journal Home Page: https://ijaems.com/
DOI: https://dx.doi.org/10.22161/ijaems.122.21

Problems of Scaling Large Language Model Inference During Simultaneous Operation of Multiple Autonomous Agents

Kapil Verma

Software Engineer, Google, Mountain View, CA, USA

Received: 21 Mar 2026; Received in revised form: 22 Apr 2026; Accepted: 25 Apr 2026; Available online: 29 Apr 2026

Abstract *– The article examines the transformation of large language models from a single-query, single-response regime to a multi-agent configuration, in which a single external stimulus generates a tree of dependent calls to the model, and analyzes the specific constraints on scaling inference. The relevance of the study is determined by the proliferation of LLM-based agents and the growing share of workloads in which not the total number of tokens but the cadence of short iterations is decisive. The objective is to identify causal bottlenecks that determine throughput and tail latencies during the concurrent operation of multiple autonomous executors. On the basis of an analytic–synthetic review of 11 sources, a framework is proposed that shifts the unit of analysis from an individual response to a chain of dependent micro-steps, interpreted as competing job classes. The scientific contribution consists of systematizing the role of the KV cache as a dynamic scarce resource, introducing the phenomenon of contextual inflation, and linking these effects to batching policies, service fairness, distributed inference, and step routing across models of different sizes. It is shown that bottlenecks in multi-agent systems shift from arithmetic performance to memory, attention-state management, stopping discipline, and context engineering, while tail and network latencies take on the character of cascading lockups; the necessity of role-dependent token budgets and carefully designed eviction and state-folding strategies is substantiated. The article is intended for researchers and engineers developing LLM-based multi-agent systems and the infrastructure for their operation.*

Keywords *–* ***large language models, multi-agent systems, inference scaling, KV cache.***

I. INTRODUCTION

In recent years, large language models have ceased to function merely as one interlocutor per query and increasingly act as the computational core of systems in which multiple autonomous software executors jointly solve a task, partitioning planning, search, verification, and answer construction into specialized roles [1]. This transition has become practical for two reasons. First, the same model can support heterogeneous cognitive functions if the input–output format is rigidly fixed and the role is imposed via context, reducing development costs compared with training separate models for each module [2]. Second, collective organization enhances robustness: errors of one executor are partially compensated by a critic, tool-based verification, and repeated refinement cycles, particularly evident in complex multi-step tasks where the priority is not elegant text but consistent decision-making [3]. This evolution from single executors to multi-agent configurations and communities is systematized in recent survey works, which emphasize that interaction among specialized roles has become the canonical engineering pattern for improving the quality and controllability of model behavior [4].

The load induced by the simultaneous operation of many autonomous executors differs qualitatively from that of an ordinary chat, where each

user query is typically completed by a single, relatively long response. In a multi-agent system, one external query triggers a cascade of internal calls to the model: short decision-making commands are interleaved with intermediate plan generation, clarifying tool calls, and repeated checks, and these chains are launched in parallel and often synchronize on shared resources [5]. As a result, not only does the average number of calls increase, but the variance in context and response length grows: alongside long inputs (action logs, tool outputs, replanning) appear short one- or two-step replies, followed again by extended text fragments. This heterogeneity makes tail latencies critical and amplifies queue-blocking effects, forcing serving systems for generative models to abandon naïve run-to-completion schemes and to seek more flexible computation-scheduling strategies [6].

But there is also a subtler difference: in chat mode, the user is mostly interested in the final result. In multi-agent mode, on the other hand, there is an individual cost per step, and since each step depends on the previous one, any one-step slowdown affects all steps. As a consequence, since the bottleneck is not in how many words are generated but in how many small steps with different contexts can be taken, high performance and low latency cannot be achieved simultaneously. This change in the unit of work, from a response to a sequence of interdependent steps, provides the motivation for further analysis of inference-scaling challenges under concurrent operation of a multitude of autonomous executors [7].

II. MATERIALS AND METHODOLOGY

The empirical material for the study comprises 11 peer-reviewed sources selected as a representative cross-section of two interrelated lines of work: (a) survey and architectural studies of multi-agent LLM configurations and their typical operating loops, and (b) systems papers on inference serving, where latency, batching, and memory management are treated as primary scaling constraints. As the theoretical foundation, surveys on LLM-based agents and multi-agent systems were used that document the shift from single dialogue to orchestration of specialized roles and iterative plan–act–observe–reflect cycles (Li et al., 2024; Xi et al., 2025; Chowa et al., 2026), as well as works that demonstrate the practical effectiveness of distributing cognitive functions via context and prompt-engineering strategies (Gozzi & Di Maio, 2024) and the robustness effects of collective verification/error smoothing in LLM-driven agents (Hu et al., 2025). Applied multi-agent case studies were not treated as benchmark sources, but as material for reconstructing a characteristic load profile: a cascade of internal calls, heterogeneity of context and response lengths, and parallel execution branches with synchronization on shared resources (Lv et al., 2025; Kim et al., 2025; Jimenez-Romero et al., 2025).

The methodology had an analytic–synthetic character and was constructed as a linkage of three procedures aimed at identifying causal bottlenecks in the problem of inference scaling under concurrent operation of multiple autonomous agents. First, a conceptual analysis of the unit of work in a multi-agent system was performed: instead of the single query, single response metric, a chain of dependent micro-steps with different urgency and cost was examined, which accentuates the importance of tail latencies and makes queue planning a central object of study (Li et al., 2024; Xi et al., 2025). Second, a comparison of systems-level mechanisms for serving LLMs was conducted from the perspective of their behavior under heterogeneous agent load, with flexible scheduling/speculative execution strategies and their potential incompatibility with short, format-strict agent iterations considered (Miao et al., 2024; Zhao & Wang, 2024). Third, a structural analysis of resource constraints was conducted through the lens of memory and attention state: the KV cache and its allocations were treated as a dynamic scarce resource that determines the upper bound on parallelism and the predictability of latencies, with the principles of PagedAttention used as a canonical model of how memory management becomes a system-forming factor in inference serving (Kwon et al., 2023).

III. RESULTS AND DISCUSSION

Multi-agent systems based on large language models are typically constructed as a composition of specialized executors that sequentially or in parallel perform different cognitive functions: one forms a plan and decomposes the goal into steps, another

executes steps and invokes external tools, a third acts as a critic and attempts to refute intermediate conclusions, a fourth retrieves relevant information from external memory or a search environment, and a fifth monitors process state and enforces constraints on time, cost, and safety. Such functional decomposition is described in recent survey studies as a stable architectural technique that replaces a universal interlocutor with a system of interacting roles, where quality is achieved not only through model parameters but also through the structure of interaction and feedback [8]. In practical scenarios, this manifests in software engineering, research workflows, simulations, and tool-use tasks, where agent behavior must be reproducible and auditable rather than merely persuasive [9].

The orchestration of such executors can be centralized, with a single dispatcher allocating steps and budgets and managing queues of calls to the model, or distributed, with executors exchanging messages and making decisions locally, coordinating actions through an interaction protocol. In both variants, an iterative plan–act–observe–reflect loop dominates, in which observation includes the results of external calls, and reflection involves replanning in light of new data and criticism. A survey on autonomous agents emphasizes that the presence of a closed loop with tools and memory turns the language model into a component that functions as a control device: it must not only generate text, but also maintain state, respond to environmental signals, and adjust strategy [8]. This is important for scaling inference because the loop generates many short control calls interleaved with rare but long episodes of generation and summarization, and also creates long-lived dialogues among executors.

The load profile under parallel agent operation is characterized by a single external query unfolding into an internal call tree: a planner may spawn several execution branches, each branch may initiate a series of tool clarifications, and the critic may generate additional verification and alternative-hypothesis queries. As a consequence, not only does the aggregate call frequency grow, but the dispersion of input and output lengths increases: short commands and probing utterances are interspersed with long contexts containing action logs and embedded tool results [10]. At the computational level, this splits into two asymmetric phases: input-context processing, in which the model iterates over all input tokens, and step-wise generation, in which each subsequent token is computed sequentially. The paradox is that multi-agent organizations often make the first phase expensive due to context growth, while simultaneously increasing sensitivity to the second phase, because latency at every short step stretches the entire decision-making loop and worsens system-level tail latency.

The primary constraints quickly shift from the raw arithmetic performance of the accelerator to memory and its bandwidth, since accelerating step-wise generation requires storing intermediate attention states in a key–value cache. In an experimental configuration on an accelerator with 40 gigabytes of video memory, it is shown that this cache can occupy more than one-third of the device memory even in typical serving regimes, and inefficient management leads to severe fragmentation and loss of usable capacity: measurements for existing systems revealed that the actually utilized memory for token states amounted to only about one-fifth to two-fifths of the allocated cache volume, which directly limits the degree of parallel processing and, accordingly, the service throughput [11]. In multi-agent mode, the effect is exacerbated by long-lived sessions and unpredictable generation length: memory grows and breathes over time, allocations become irregular, and memory bandwidth increasingly becomes the limiting factor, because step-wise generation forces the system to access large state structures repeatedly, and any unfortunate queue-scheduling decision amplifies lockups and latency spikes. The multi-agent LLM system challenges are systematized in Figure 1.

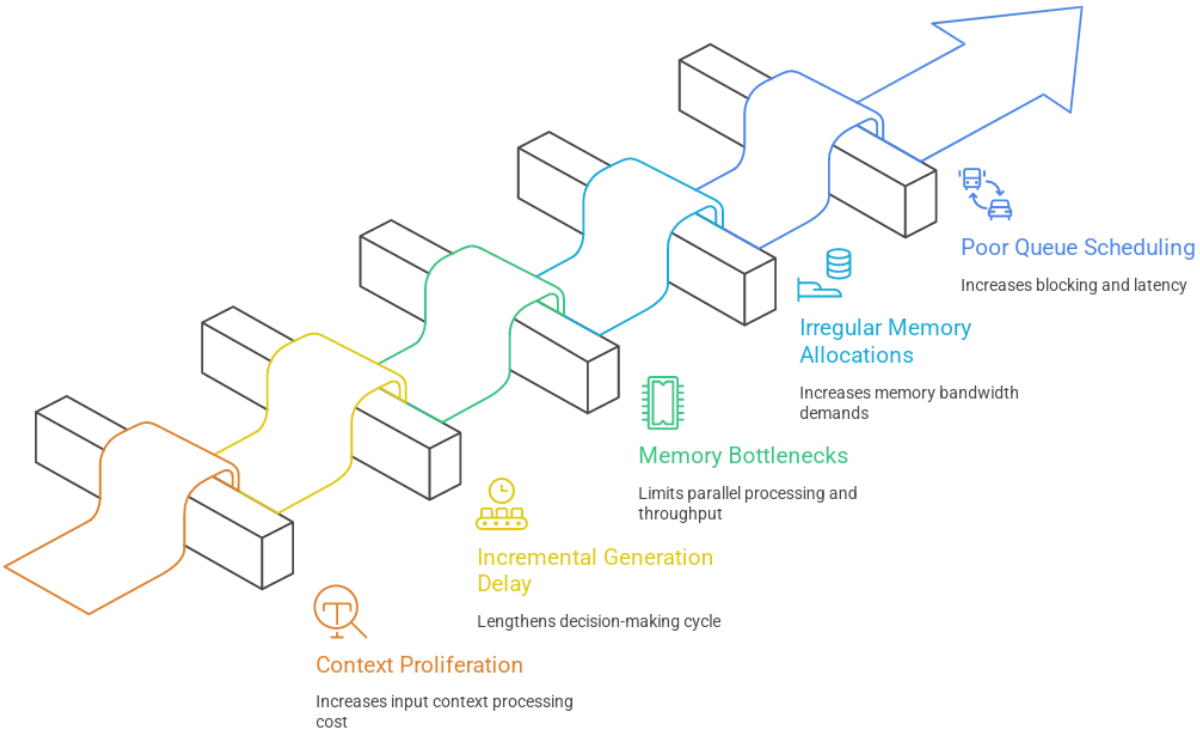

Fig. 1. Multi-Agent LLM System Challenges

With the parallel execution of numerous executors, model inference becomes a dispatching problem in which heterogeneous requests compete on a single accelerator, while the objective is twofold: to maximize aggregate throughput and to maintain step latency within tight bounds. Batching requests increases the efficiency of compute-unit utilization, but the price of this efficiency manifests itself as waiting time: short steps are forced to accumulate companions, thereby elongating the closed decision-making cycles of agents. As a result, the same batching mechanism can improve average performance while degrading perceived system reactivity, especially when the agent logic is structured as a chain of dependent steps, where each subsequent step is launched only after the previous response has been received.

A critical subtlety is that input-context processing and step-wise generation behave as distinct job classes and require different batching rules. Input processing benefits from large batches because computations proceed as dense matrix operations that fully utilize the accelerator, whereas step-wise generation is sensitive to waiting, since it consists of many short iterations that accumulate latency across steps. If these modes are mixed without differentiation, queues form in which heavy requests with long inputs block the flow of small iterations, and the system experiences a head-of-line blocking effect: fast jobs idle behind slow ones, even though the resource could have served them in a different order. In the limiting case, some executors begin to starve because their small requests are constantly displaced by heavier neighbors or by more active agents that generate a higher volume of calls.

A separate issue is fairness, because in a multi-agent environment, requests are incomparable in both cost and importance. One executor may generate long contexts and long responses, consuming memory and time; another may issue short control commands that are critical to the entire system; a third may periodically initiate monitoring or verification steps in which the cost of error exceeds the cost of delay. Therefore, simple fairness by number of requests fails: policies are required that account for token budgets, the cost of memory for state, step urgency, and the agent's role in the overall process. Without such policies, the system either becomes unaffordable or loses robustness as critical decision branches begin to experience systematic delays.

Growth in context length amplifies all of the aforementioned effects and introduces the

phenomenon of contextual inflation, which is particularly characteristic of autonomous executors. During operation, action logs, results of external calls, intermediate plan versions, and self-critiques accumulate, and there is a tendency to feed all of this back into the model for reliability. However, every redundant input fragment increases processing costs and inflates the state required for generation, while also bringing the system closer to the context-window limit. Once the window overflows, the developer must either discard data and risk losing important decision justifications or compress them and risk semantic distortion; in addition, an overly saturated input degrades quality because the model diffuses its attention and more frequently confuses current facts with obsolete traces of previous steps. Consequently, compression and deduplication become prerequisites for scaling rather than cosmetic refinements: the system must be able to transform a long history into a compact state, store knowledge outside the primary input, and retrieve it only when necessary; otherwise, computational resources will inevitably be consumed by textual noise.

The use of external tools disrupts the computational flow and violates the assumption of a smooth queue. Pauses for external services, which exhibit their own latencies and failures, appear between agent steps; upon completion of such calls, many executors often return to the model simultaneously, creating load spikes. At the infrastructure level, a state-management dilemma arises: it is possible to retain intermediate generation state to resume the dialogue quickly, but then memory requirements grow, and long-lived session management becomes more complex; alternatively, state can be released and the input reprocessed from scratch, at the cost of increased compute and context-reconstruction latency. In multi-agent systems, this dilemma becomes especially acute because pauses and spikes are the norm, and the choice of strategy directly determines whether the system remains resilient to peak loads and can preserve reactivity without uncontrolled cost growth. Challenges in parallel model execution are shown in Figure 2.

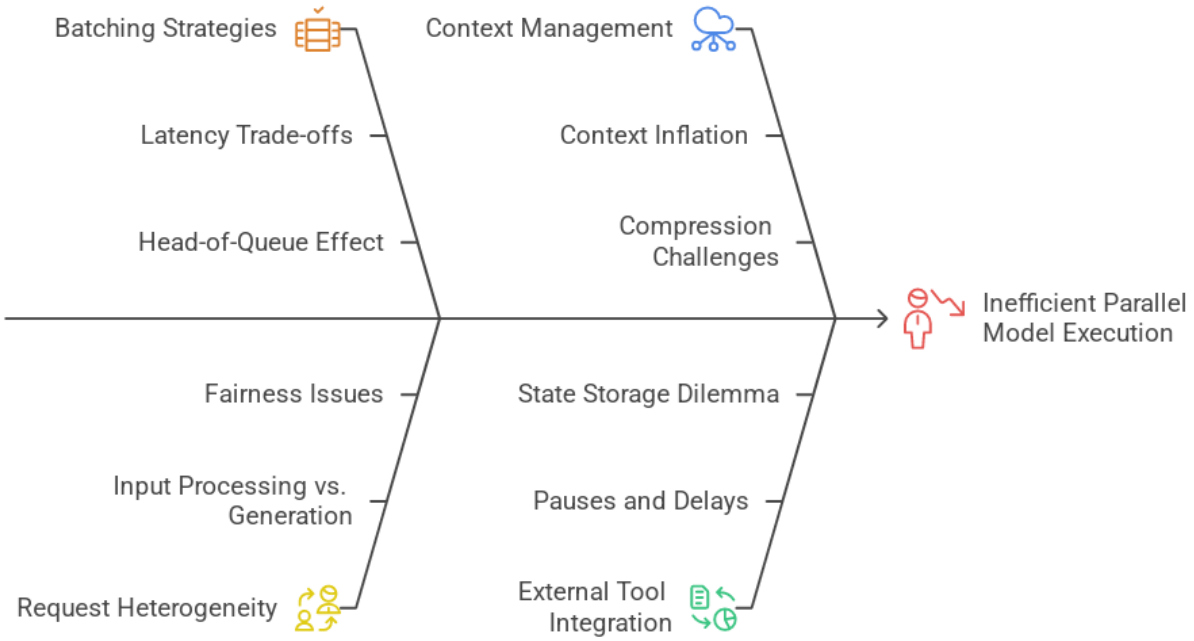

Fig. 2. Challenges in Parallel Model Execution

When the parallel operation of executors saturates memory and the queue on a single accelerator, a natural step is to distribute computation across multiple devices and nodes. However, in distributed mode, inference ceases to be a local operation and becomes an alternation of compute and communication phases, in which intermediate representations must be repeatedly transmitted among participants. Under layer-wise partitioning, some nodes are forced to wait for neighbors to complete, and

under parameter partitioning, each step requires coordination of partial results, with the cost of this coordination growing as the number of participants increases. For multi-agent systems, this is particularly problematic because they generate many short steps: the finer the step granularity, the higher the fraction of time spent on coordination rather than useful computation, and the harder it becomes to hide communication behind computation.

The network contributes not only to increased average latency but also to increased variability, which manifests as tail latencies and introduces instability into closed-loop control cycles. Even if the mean response time is acceptable, rare but long delays break synchronization among executors: one agent waits for the critic's confirmation, the critic waits for a tool result, and the dispatcher holds resources, unable to determine whether a branch will complete. In such a system, tail latency becomes not a mere statistic but a mechanism of cascading slowdown, in which an individual stall escalates into a series of lockups, reinforcing queueing effects and increasing cost due to retries and context expansion. The problem is exacerbated by load spikes: agent systems often exhibit staged behavior, and when several branches simultaneously return from external calls, the distributed infrastructure experiences a synchronous influx of requests that is poorly absorbed because scaling computation requires warm-up, state distribution, and queue stabilization.

Against this backdrop, model-level optimizations are treated as a means to alleviate pressure on compute and memory, but they inevitably embed a trade-off between speed and quality. Reducing parameter precision frees memory and increases parallel-processing density; however, in multi-step chains, even minor distortions can accumulate: an early inaccuracy in the plan leads to an incorrect tool choice, after which an erroneous result is recorded in the context, and the critic then evaluates an already distorted picture and reinforces false confidence. Thus, a gain in speed may turn into an increase in the number of iterations and in total cost, because the agent system compensates for quality degradation with additional refinement and self-verification cycles.

A separate class of methods attempts to accelerate generation by predicting fragments of the answer more quickly and then confirming them via more precise computation. This yields benefits when responses are long and predictable, but becomes less meaningful when agent steps are short, frequently interrupted by tool calls, and require strict adherence to format. In a multi-agent environment, many responses are control commands or compact conclusions, where the savings from accelerated generation are minimal, while the overhead of verification and correction becomes noticeable. Moreover, such methods complicate queue planning because they introduce yet another source of heterogeneity: some requests are accelerated, others are not, and the dispatcher must anticipate this in advance. Otherwise, there will be no net benefit.

For this reason, routing requests among models of different sizes is increasingly employed, with simple steps handled by lighter models and complex, high-stakes decisions delegated to more powerful ones. At the architectural level, this resembles the division of labor among executors but is shifted within the computational layer: the system attempts not only to distribute agent roles but also to assign each step an appropriate computational depth. Nonetheless, this strategy requires a careful complexity criterion; otherwise, routing errors become systematic: a light model may take over a step that requires precision and thereby launch a correction cascade, or, conversely, a heavy model may expend resources on trivial actions and worsen latency for others. Ultimately, model-level optimizations cannot be considered in isolation from agent dynamics: their effect is determined by how they alter the number of iterations, the shape of the query tree, and the system's robustness to tail latencies in a distributed environment. The cycle of distributed agent system optimization is illustrated in Figure 3.

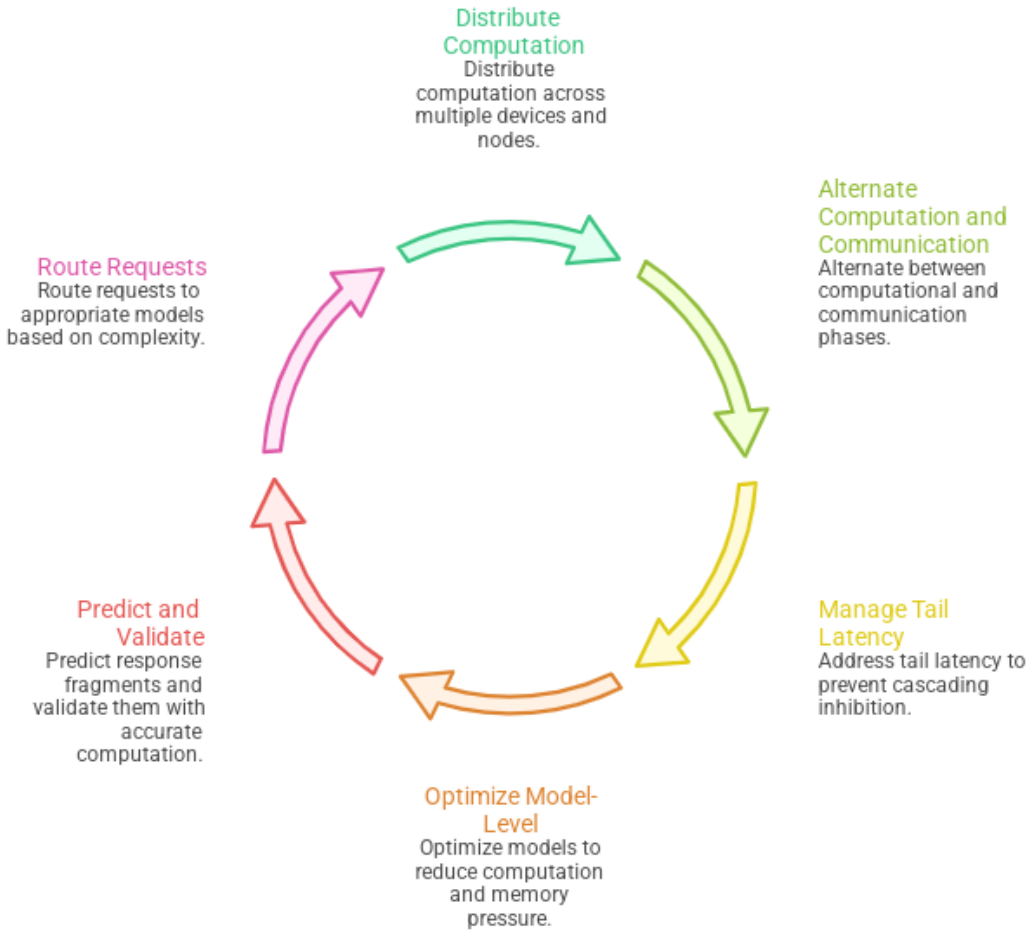

Fig. 3. Cycle of Distributed Agent System Optimization

Practice shows that scaling a multi-agent system begins not with accelerators but with stopping discipline. If executors are allowed to refine plans indefinitely, re-check themselves, and spawn new branches, any infrastructure improvement will be consumed by growth in the number of steps. It is therefore useful to predefine limits on the number of iterations and on the volume of text that each executor may add to the shared state, and these limits should be role-dependent: planners and critics are dangerous precisely because they can inflate discussion, whereas action executors more often require short commands. It is important that the constraints be not only hard but also semantic: once its budget is exhausted, an agent must switch to result-folding mode or request an external signal. Otherwise, the system will simulate activity, accumulating context and latency without commensurate quality gains.

Memory policies must take into account that attention state is a resource no less scarce than compute and, in a multi-agent environment, also temporally unstable. It is impossible to keep all sessions warm and simultaneously expect predictable latency: some states must inevitably be treated as cold and evicted according to rules that minimize damage to control chains. Good eviction is not reducible to least-recently-used because an agent may return after a pause in an external tool and prove critical for completing an entire branch; therefore, priority should be determined by proximity to a decision point, expected continuation length, and the cost of reconstruction. Ideally, the system should distinguish in advance between sessions for which it is advantageous to preserve state and sessions that are cheaper to reconstruct from a compact description; otherwise, memory will be occupied by history that no longer influences future steps.

Context engineering in multi-agent systems must be organized as knowledge management rather than transcript accumulation. However, for action

logs, tool output, and intermediate plans, it can pay off to only keep information useful for the next decision step. Thus, deduplicating and keeping only canonical facts, constraints, and verifiable facts can prevent the remainder of the search tree from becoming overwhelming. The context for a module includes the current goal, current plan, key observations, tool status, and a brief decision history. Detailed traces should be stored only for loading on request. This reduces the effort required to process input, reduces the risk of corrupting the signal of interest with noise, and makes the system's behavior more repeatable, as each step is based on a concise representation.

The most typical errors arise from a desire to hedge with text. Overly long logs turn the context into a landfill, where the model begins to confuse causes and effects and to devote attention to details unrelated to the current step. Another antipattern is excessive reflexive cycling, where the critic and planner repeatedly discuss the quality of their reasoning, creating an illusion of reliability, but incurring high latency and cost, and degrading the final result through the accumulation of confluent formulations. Finally, loose formats allow agents to write whatever they want. Because there is no regularity to the content, the system cannot collapse state, extract facts, or queue requests. Any infrastructure-level optimization has to navigate the erratic growth of tokens and requests.

IV. CONCLUSION

Multi-agent deployment of large language models crystallizes a shift from the single-query, single-response paradigm to a regime in which a single external stimulus unfolds into an internal call tree, heterogeneous in context and response length, and coupled by stepwise dependencies. In this configuration, performance is no longer measured solely by the total number of tokens: the decisive factor becomes the rate of short iterations under high input variability, because the latency of each micro-step stretches the entire plan–act–observe–reflect loop, and tail latencies transform from a statistical artifact into a systemic blocking mechanism. From this follows the central engineering challenge of scaling: the need to maintain high compute utilization and predictable reactivity simultaneously, despite the fact that batching, advantageous for dense input processing, can undermine step-wise generation latency and provoke starvation of lightweight but critical requests in the vicinity of heavy contexts.

At the resource level, the primary bottleneck quickly becomes not only compute units but also memory, primarily due to the storage of attention states in a KV cache, which, in typical regimes, can occupy a significant share of video memory and suffer from fragmentation, directly limiting parallelism and service throughput. Attempts to escape local limitations by distributing workloads across devices introduce communication phases and network variability, and in multi-agent dynamics with numerous short steps, coordination overhead and rare but long delays begin to disrupt branch synchronization in a cascading fashion. Consequently, model- and infrastructure-level optimizations cannot be evaluated apart from agent dynamics: acceleration or compression inevitably collides with error accumulation along the chain and the risk of increasing iteration counts, while robustness hinges on stopping discipline, meaningful budgets, state-eviction policies, and context engineering in which memory is managed as knowledge rather than as an unbounded transcript.

REFERENCES

[1] X. Li, S. Wang, S. Zeng, Y. Wu, and Y. Yang, "A survey on LLM-based multi-agent systems: workflow, infrastructure, and challenges," *Vicinagearth*, vol. 1, no. 9, Oct. 2024, doi: https://doi.org/10.1007/s44336-024-00009-2.

[2] M. Gozzi and F. Di Maio, "Comparative Analysis of Prompt Strategies for Large Language Models: Single-Task vs. Multitask Prompts," *Electronics*, vol. 13, no. 23, p. 4712, Nov. 2024, doi: https://doi.org/10.3390/electronics13234712.

[3] J. Hu, Y. Dong, Z. Ding, and X. Huang, "Enhancing robustness of LLM-driven multi-agent systems through randomized smoothing," *Chinese Journal of Aeronautics*, p. 103779, Aug. 2025, doi: https://doi.org/10.1016/j.cja.2025.103779.

[4] Z. Xi *et al.*, "The rise and potential of large language model-based agents: a survey," *Science China Information Sciences*, vol. 68, no. 2, p. 121101, Jan. 2025, doi: https://doi.org/10.1007/s11432-024-4222-0.

[5] W. Lv, Q. Cao, and X. Liu, "A multi-agent classical Chinese translation method based on large language models," *Scientific Reports*, vol. 15, p. 40160, Nov. 2025, doi: https://doi.org/10.1038/s41598-025-23904-0.

[6] X. Miao *et al.*, "SpotServe: Serving Generative Large Language Models on Preemptible Instances," *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 1112–1127, Apr. 2024, doi: https://doi.org/10.1145/3620665.3640411.

[7] Y. Zhao and J. Wang, "ALISE: Accelerating Large Language Model Serving with Speculative Scheduling," *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9, Oct. 2024, doi: https://doi.org/10.1145/3676536.3676659.

[8] S. S. Chowa *et al.*, "From language to action: a review of large language models as autonomous agents and tool users," *Artificial Intelligence Review*, vol. 59, no. 2, Jan. 2026, doi: https://doi.org/10.1007/s10462-025-11471-9.

[9] C. Jimenez-Romero, A. Yegenoglu, and C. Blum, "Multi-agent systems powered by large language models: applications in swarm intelligence," *Frontiers in Artificial Intelligence*, vol. 8, May 2025, doi: https://doi.org/10.3389/frai.2025.1593017.

[10] S. Kim, J. Ha, H. Lee, S. Park, and S. Cho, "Human-guided collective LLM intelligence for strategic planning via two-stage information retrieval," *Information Processing & Management*, vol. 63, no. 1, p. 104288, Jul. 2025, doi: https://doi.org/10.1016/j.ipm.2025.104288.

[11] W. Kwon *et al.*, "Efficient Memory Management for Large Language Model Serving with PagedAttention," *Proceedings of SOSP '23*, pp. 611–626, Oct. 2023, doi: https://doi.org/10.1145/3600006.3613165.