Proceedings of the Thirty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2025)

DynTaskMAS: A Dynamic Task Graph-driven Framework for Asynchronous and Parallel LLM-based Multi-Agent Systems

Junwei Yu[1*], **Yepeng Ding**[2*], **Hiroyuki Sato**[1,3]
[1]The University of Tokyo
[2]Hiroshima University
[3]National Institute of Informatics
yujw@satolab.itc.u-tokyo.ac.jp, yepengd@acm.org, schuko@nii.ac.jp

Abstract

The emergence of Large Language Models (LLMs) in Multi-Agent Systems (MAS) has opened new possibilities for artificial intelligence, yet current implementations face significant challenges in resource management, task coordination, and system efficiency. While existing frameworks demonstrate the potential of LLM-based agents in collaborative problem-solving, they often lack sophisticated mechanisms for parallel execution and dynamic task management. This paper introduces DynTaskMAS, a novel framework that orchestrates asynchronous and parallel operations in LLM-based MAS through dynamic task graphs. The framework features four key innovations: (1) a Dynamic Task Graph Generator that intelligently decomposes complex tasks while maintaining logical dependencies, (2) an Asynchronous Parallel Execution Engine that optimizes resource utilization through efficient task scheduling, (3) a Semantic-Aware Context Management System that enables efficient information sharing among agents, and (4) an Adaptive Workflow Manager that dynamically optimizes system performance. Experimental evaluations demonstrate that DynTaskMAS achieves significant improvements over traditional approaches: a 21-33% reduction in execution time across task complexities (with higher gains for more complex tasks), a 35.4% improvement in resource utilization (from 65% to 88%), and near-linear throughput scaling up to 16 concurrent agents (3.47× improvement for 4× agents). Our framework establishes a foundation for building scalable, high-performance LLM-based multi-agent systems capable of handling complex, dynamic tasks efficiently.

Introduction

The rapid advancement of LLMs has revolutionized the landscape of artificial intelligence, demonstrating unprecedented capabilities in natural language understanding and generation (Brown 2020). These models have shown remarkable proficiency in tasks ranging from text completion to complex reasoning, opening new avenues for intelligent system design. Concurrently, the concept of MAS has gained significant traction, offering a distributed approach to problem-solving that leverages the collective intelligence of multiple entities (Liu et al. 2023; Khattab et al. 2023; Gao et al. 2024; Park et al. 2023; Li et al. 2023; Yang, Yue, and He 2023). The convergence of these two paradigms presents a promising frontier in AI research, with the potential to address increasingly complex and dynamic challenges across various domains.

LLM-driven multi-agent systems have exhibited substantial potential in tackling intricate problems that require diverse expertise and collaborative reasoning. However, current research in this field has predominantly focused on simplified architectures with single topology structures and straightforward task flows. While these approaches have yielded valuable insights, they fall short in addressing the multifaceted demands of real-world applications, where task complexity and environmental dynamics can vary significantly.

As the complexity of tasks escalates, traditional MAS architectures face several critical challenges. Firstly, the task decomposition process becomes increasingly intricate, requiring more sophisticated mechanisms to break down complex problems into manageable subtasks while maintaining logical coherence. Secondly, the parallel processing capabilities of existing systems are often underutilized, leading to inefficiencies in resource allocation and execution time. Thirdly, context management across multiple agents becomes exponentially more challenging as the number of interactions and the volume of shared information grow.

To address these limitations, we propose DynTaskMAS, a Dynamic Task Graph-driven Framework for Asynchronous and Parallel LLM-based Multi-Agent Systems. By placing a dynamic task graph, DynTaskMAS enables flexible task decomposition and efficient parallel execution, effectively overcoming the aforementioned challenges. The key contributions of this work are as follows:

1. This work proposes a novel framework for dynamic task orchestration in LLM-based MAS. The approach introduces new principles for decomposing complex language tasks while maintaining semantic coherence and causal dependencies.
2. The research advances the understanding of parallel execution patterns in LLM-based systems through new abstractions for managing asynchronous agent interactions. This theoretical foundation enables more efficient resource utilization while preserving the complex reasoning capabilities inherent to language models.

[*]These authors contributed equally.

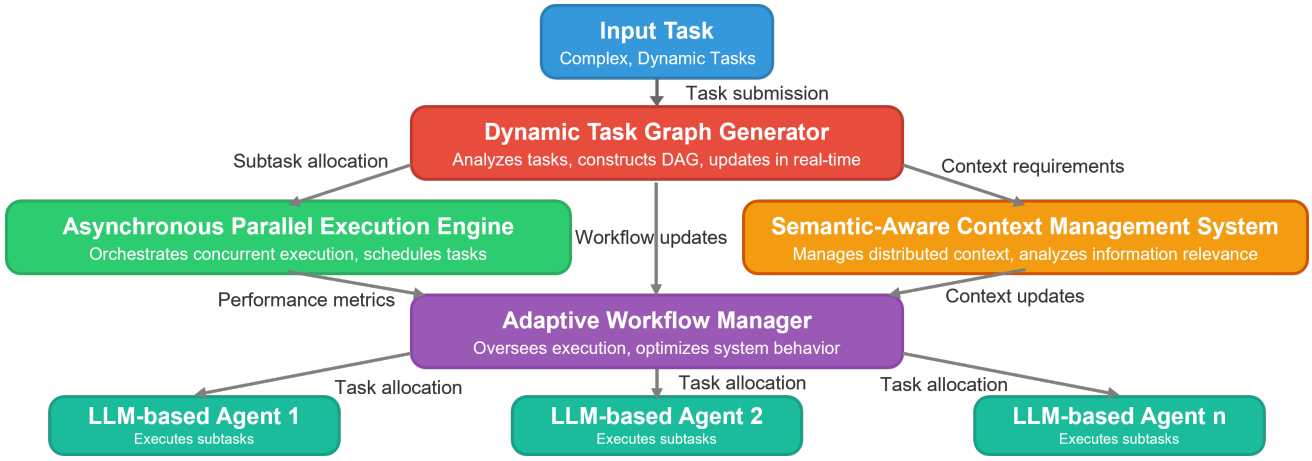

Figure 1: **The Overview of DynTaskMAS Framework.** The diagram depicts the architecture of the DynTaskMAS framework as a multi-layered structure characterized by bidirectional information flow. At the top, the "Input Task" node serves as the entry point for complex tasks. Below it, DTGG acts as a core component, continuously analyzing and updating task graphs, with arrows connecting to the input task to signify this iterative process. The third layer comprises two parallel modules: APEE and SACMS. Both are linked to DTGG above and AWM below, reflecting their interdependent functions. Positioned in the fourth layer, AWM maintains bidirectional connections with all upper components, underscoring its role in managing and optimizing workflows. At the base, multiple "LLM-based Agent" nodes are connected to APEE and SACMS, illustrating the distributed execution of tasks and continuous exchange of information.

3. A systematic approach to context management in multi-agent LLM systems addresses the fundamental challenge of maintaining semantic relevance across distributed agents. The methodology introduces new ways to balance information sharing with computational efficiency, establishing principles for scalable LLM-based architectures.
4. Experimental results demonstrate substantial improvements over traditional approaches: 21-33% reduction in execution time, 35.4% increase in resource utilization, and near-linear throughput scaling up to 16 concurrent agents.

Background

Planning with LLMs

Recent studies have explored the capacity of LLMs to perform zero-shot planning, converting high-level natural language tasks into actionable steps without additional training, albeit with noted challenges in precision mapping to executable actions (Huang et al. 2022). Other research has focused on enhancing LLMs' reasoning capabilities through structured prompting techniques, eliciting more deliberate and logical reasoning processes (Fagbohun, Harrison, and Dereventsov 2024). The integration of LLMs with robotic affordances has been demonstrated to enable complex instruction following in real-world scenarios, showcasing the potential for LLMs to ground language in physical interactions (Li et al. 2024). Further advancements have been made in few-shot grounded planning, where LLMs are employed to generate plans for embodied agents within visual environments. Interactive planning methods have been developed to facilitate open-world multi-tasking, leveraging LLMs to describe, explain, plan, and select actions (Rasal and Hauer 2024). Additionally, frameworks that empower LLMs with optimal planning proficiency have been proposed, aiming to solve planning problems articulated in natural language (Xiong et al. 2024). Problem-solving and decision-making capabilities of LLMs have been augmented through novel frameworks designed to mimic human strategic planning (Yao et al. 2024). Collectively, these contributions underscore the evolving role of LLMs in planning and reasoning, highlighting their potential to enhance embodied agent performance across a spectrum of tasks.

Building upon these contributions, which have significantly advanced the understanding of LLMs in planning and reasoning, there is a notable gap in research focusing on the operational efficiency and practical considerations of resource allocation, particularly in the context of real-time GPU resource management. While these studies have underscored the evolving role of LLMs in enhancing embodied agent performance, they have largely overlooked the challenges associated with the asynchronous and parallel execution of tasks within multi-agent systems.

Multi-Agent System

In the field of multi-agent systems utilizing large language models, several notable approaches have emerged. AutoGPT demonstrates an autonomous system where agents collaboratively achieve goals through iterative planning, execution, and evaluation cycles (Yang, Yue, and He 2023). Building on this concept, MetaGPT emulates a startup team structure, employing role-specific agents to tackle complex tasks like software development and project planning (Hong et al. 2023). Camel (Collaborative Agents for Multi-agent Environment Learning) further enhances collaborative problem-

solving by simulating scenarios that require communication, knowledge sharing, and collective decision-making among language models (Li et al. 2023). Stanford University's Generative Agents take a different approach, focusing on simulating lifelike behavior in digital environments by creating personas with memory systems and adaptive behaviors (Park et al. 2023). While these multi-agent systems showcase innovative approaches to collaboration and task execution, they have not yet provided superior solutions for parallel and serial processing challenges, indicating an area for potential future research and development in the field.

DynTaskMAS

DynTaskMAS is a novel framework designed to enhance the efficiency and adaptability of LLM-based multi-agent systems in handling complex, dynamic tasks. The architecture comprises four primary components that work in concert to achieve flexible task management and optimized resource utilization:

Dynamic Task Graph Generator (DTGG): This component analyzes incoming tasks and automatically constructs a directed acyclic graph (DAG) representing subtasks and their interdependencies. The DTGG continuously updates the graph as new information becomes available or task requirements change, ensuring adaptability to dynamic environments.

Asynchronous Parallel Execution Engine (APEE): The APEE orchestrates the concurrent execution of subtasks across multiple LLM-based agents. It employs sophisticated scheduling algorithms to maximize parallelism while respecting task dependencies defined in the dynamic task graph.

Semantic-Aware Context Management System (SACMS): This subsystem facilitates efficient information sharing among agents by maintaining a hierarchical, distributed context repository. The SACMS employs semantic analysis to determine the relevance of information, ensuring agents have access to pertinent data without unnecessary overhead.

Adaptive Workflow Manager (AWM): The AWM oversees the overall execution process, dynamically adjusting workflows based on real-time performance metrics and environmental changes. It interfaces with all other components to optimize system behavior and resource allocation.

In the illustrated Figure 1, these components interact through a central coordination mechanism that ensures coherent system operation. The modular design of DynTaskMAS allows for scalability and easy integration of additional agents or task types, making it adaptable to various application domains.

Dynamic Task Graph Generator

The Dynamic Task Graph Generator is a crucial component of DynTaskMAS, responsible for decomposing complex tasks into manageable subtasks and representing their dependencies as a DAG. The DTGG continuously updates this graph based on new information and changing task requirements (Kwok and Ahmad 1999).

Graph Structural Composition: Let $T = t_1, t_2, ..., t_n$ be the set of all tasks in the system. The dynamic task graph G is defined as:

$$G = (V, E, W) \tag{1}$$

where $V = \{v_1, v_2, \ldots, v_m\}$ is the set of vertices representing subtasks, $E \subseteq V \times V$ is the set of edges indicating dependencies, and $W : E \to \mathbb{R}^+$ is a weight function assigning positive real values to the edges.

The DTGG employs a recursive decomposition algorithm to break down complex tasks. For each task $t_i \in T$, we define a decomposition function D:

$$D(t_i) = s_{i1}, s_{i2}, ..., s_{ik} \tag{2}$$

where s_{ij} are subtasks of t_i. The decomposition continues until a predefined granularity level is reached.

The weight of an edge $(v_i, v_j) \in E$ is calculated based on the estimated computational complexity and data dependency between subtasks:

$$W(v_i, v_j) = \alpha \cdot C(v_j) + \beta \cdot I(v_i, v_j) \tag{3}$$

where $C(v_j)$ denotes the estimated computational complexity of subtask v_j, $I(v_i, v_j)$ represents the context transfer time managed by SACMS from v_i to v_j, and α, β are balancing coefficients. For optimal parameter selection, we recommend:

- $\alpha = \frac{1}{T_c}$, where T_c is the average computation time per complexity unit
- $\beta = \frac{1}{T_t}$, where T_t is the average context transfer time per unit of information

This normalization ensures that both computational complexity and context transfer time contribute proportionally to the edge weight, enabling more accurate task scheduling decisions. The ratio $\frac{\alpha}{\beta}$ should be adjusted based on the system's relative speeds of computation versus context transfer, typically ranging from 0.5 to 2.0 depending on the specific hardware configuration and network conditions.

Control of Cyclic Dependencies: To manage the complexity of reflection cycles in the DTGG, it is essential to distinguish between true cyclic dependencies and apparent cyclic structures. In LLM-based MAS, where each prompt functions as an agent, genuine cyclic dependencies primarily manifest in reflection processes, where an agent evaluates and refines its output. Other apparent cycles, such as iterative refinement or progressive enhancement, are essentially linear task sequences with clear hierarchical dependencies.

To prevent infinite loops or excessive processing in reflection cycles, we implement a maximum iteration threshold N (typically $N \leq 3$). The reflection terminates when either:

- The quality assessment meets the predetermined threshold
- The number of reflection cycles reaches N
- The improvement between successive iterations falls below a minimum threshold ε

The DTGG continuously updates the task graph based on new information and task progress. Let G_t be the graph at

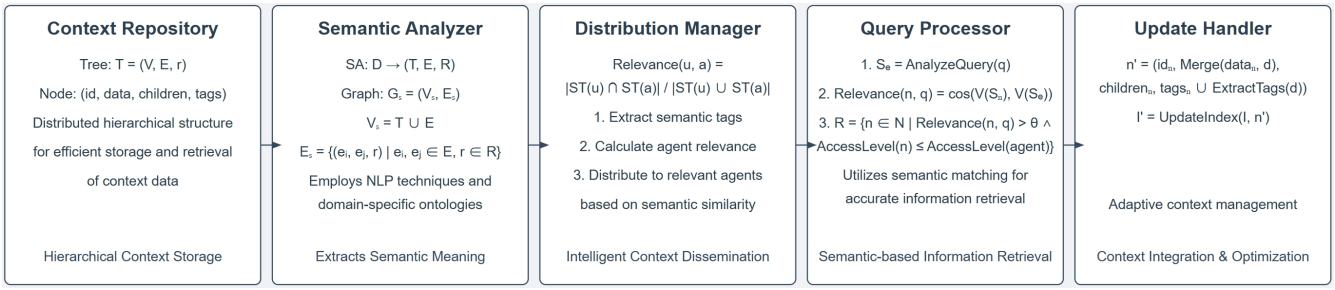

Figure 2: **The architecture of Semantic-Aware Context Management System.** The SACMS architecture comprises five core components: (1) the Context Repository, a distributed, hierarchical data store designed for efficient storage and retrieval of contextual information; (2) the Semantic Analyzer, which employs advanced natural language processing and domain-specific ontologies to extract meaningful semantic tags and relationships; (3) the Context Distribution Manager, responsible for the efficient dissemination of relevant contextual information to agents based on task requirements and semantic relevance; (4) the Query Processor, which utilizes semantic matching to process context retrieval requests and deliver the most pertinent information; and (5) the Update Handler, which integrates new or updated context data into the repository while maintaining the semantic index. Together, these components enable intelligent, context-aware task execution within the DynTaskMAS system.

Algorithm 1: Dynamic Task Graph Generator

function UPDATETASKGRAPH(G, T_{new}, Δ)
  for each t_i in T_{new} **do**
    $S_i \leftarrow$ DECOMPOSETASK(t_i)
    $V \leftarrow V \cup S_i$
    for each s_{ij}, s_{ik} in S_i where $j < k$ **do**
      $E \leftarrow E \cup (s_{ij}, s_{ik})$
      $W(s_{ij}, s_{ik}) \leftarrow$ CALCWEIGHT(s_{ij}, s_{ik})
  for each change δ in Δ **do**
    $G \leftarrow$ APPLYCHANGE(G, δ)
  return G
function DECOMPOSETASK(t)
  if ISATOMICTASK(t) **then**
    return t
  else
    $S \leftarrow \emptyset$
    for each subtask s of t **do**
      $S \leftarrow S \cup$ DECOMPOSETASK(s)
    return S
function CALCWEIGHT(v_i, v_j)
  $C_j \leftarrow$ ESTIMATECOMPLEXITY(v_j)
  $I_{ij} \leftarrow$ ESTIMATEINFORMATIONTRANSFER(v_i, v_j)
  return $\alpha \cdot C_j + \beta \cdot I_{ij}$

time t, and Δ_t be the set of changes at time t. The update function U is defined as:

$$G_{t+1} = U(G_t, \Delta_t) \tag{4}$$

The pseudocode of the main DTGG algorithm is provided in Algorithm 1. DTGG Receives complex tasks and updates from the system's input layer. The DTGG's output, a continuously updated task graph, serves as the foundation for efficient task distribution and execution in the DynTaskMAS framework. Its ability to dynamically adjust to changing conditions ensures the system's adaptability in complex, evolving environments.

Asynchronous Parallel Execution Engine

The Asynchronous Parallel Execution Engine is a critical component of DynTaskMAS, responsible for efficiently scheduling and executing tasks across multiple LLM-based agents. It leverages the dynamic task graph generated by the DTGG to maximize parallelism while respecting task dependencies. The APEE consists of the following subcomponents: Task Scheduler, Execution Queue Manager, Agent Pool Manager, Load Balancer and Asynchronous Communication Handler.

Task Scheduler: The Task Scheduler is responsible for determining the execution order of tasks based on the dynamic task graph. It employs a priority-based scheduling algorithm that considers task dependencies, estimated execution time, and system load. Let $G = (V, E, W)$ be the current task graph. For each task $v_i \in V$, we define its priority $P(v_i)$ as:

$$P(v_i) = \frac{C(v_i)}{\max_{v_j \in \text{Succ}(v_i)}(W(v_i, v_j) + P(v_j))} \tag{5}$$

where $C(v_i)$ is the estimated computational complexity of task v_i, $\text{Succ}(v_i)$ is the set of immediate successors of v_i in the graph, and $W(v_i, v_j)$ is the weight of the edge (v_i, v_j).

Execution Queue Manager: The Execution Queue Manager maintains a priority queue of ready-to-execute tasks. It continuously updates the queue based on the Task Scheduler's output and the current system state.

Agent Pool Manager and Load Balancer: The Agent Pool Manager and Load Balancer work together to efficiently manage and distribute tasks among LLM-based agents. The Agent Pool Manager oversees the pool of available agents, tracking their status, capabilities, and workload. It provides the necessary interface for the Load Balancer, which ensures optimal task distribution based on task priorities, agent capabilities, and current system load. Together, they coordinate to allocate tasks to the most suitable agents, maintaining efficient system performance.

Asynchronous Communication Handler: The Asynchronous Communication Handler manages the non-

Algorithm 2: Execution Queue Manager

function UPDATEEXECUTIONQUEUE(G, Q)
  $R \leftarrow v \in V : \text{Pred}(v) = \emptyset$ ▷ Ready tasks
  for each $v \in R$ **do**
    $priority \leftarrow$ CALCPRIORITY(v, G)
    Q.ENQUEUE($v, priority$)
  return Q
function CALCPRIORITY(v, G)
  if Succ(v) $= \emptyset$ **then**
    return $C(v)$
  else
    return $\frac{C(v)}{\max_{u \in \text{Succ}(v)}(W(v,u)+\text{CALCPRIORITY}(u,G))}$

blocking communication between the APEE and the LLM-based agents. It uses an event-driven architecture (task completion notifications, agent availability updates, and task failure reports) to handle task assignments, status updates, and result collections to ensure high throughput and responsiveness.

By leveraging the dynamic task graph and employing sophisticated scheduling and load balancing algorithms, the APEE enables DynTaskMAS to achieve high levels of parallelism and efficiency in executing complex, interdependent tasks across multiple LLM-based agents.

Semantic-Aware Context Management System

The SACMS is a pivotal component of DynTaskMAS, designed to efficiently manage and distribute contextual information among LLM-based agents. Figure 2 depicts the architecture and key algorithms of SACMS, highlighting its role in enabling intelligent and context-aware task execution. By leveraging advanced semantic analysis techniques, distributed architecture, and adaptive mechanisms, SACMS enables LLM-based agents to access and utilize contextual information effectively, thereby enhancing the overall performance and adaptability of the system.

Context Repository: The Context Repository serves as the central data store for contextual information. It is implemented as a distributed, hierarchical structure to facilitate efficient storage and retrieval of context data.

The repository is organized as a forest of trees, where each tree represents a distinct context domain. This structure allows for natural representation of hierarchical relationships within contexts.

$$\text{ContextForest} = T_1, T_2, ..., T_n \tag{6}$$

where each T_i is a context tree defined as:

$$T_i = (V_i, E_i, r_i) \tag{7}$$

with V_i being the set of nodes, E_i the set of edges, and r_i the root node of the tree.

Each node in the context tree is defined as follows:

$$\text{ContextNode} = (\textit{id}, \textit{data}, \textit{children}, \textit{semanticTags}) \tag{8}$$

where *id* is a unique identifier, *data* contains the actual context information, *children* is a set of child nodes, and *semanticTags* is a set of semantic labels associated with the node.

Algorithm 3: Context Distribution

function DISTRIBUTECONTEXT(*update*, *agents*)
  $R \leftarrow \emptyset$ ▷ Relevant agents set
  tags $\leftarrow$ EXTRACTSEMANTICTAGS(*update*)
  for *agent* $\in$ *agents* **do**
    agentTags $\leftarrow$ GETTAGS(*agent*)
    if RELEVANCE(*tags*, *agentTags*) $> \theta$ **then**
      $R \leftarrow R \cup \{\textit{agent}\}$
  for *agent* $\in R$ **do**
    SENDUPDATE(*agent*, *update*)

Semantic Analyzer: The Semantic Analyzer is responsible for processing contextual information to extract meaningful semantic tags and relationships. It employs specialized LLMs and domain-specific ontologies to achieve this.

The semantic analysis process can be formalized as follows:

$$D \rightarrow (T, E, R) \tag{9}$$

where D is the input context data, T is the set of extracted tags, E is the set of identified entities, and R is the set of relationships between entities.

The analyzer constructs a semantic graph $G_s = (V_s, E_s)$ where:

$$V_s = T \cup E \tag{10}$$

$$E_s = \{(e_i, e_j, r) \mid e_i, e_j \in E, r \in R\} \tag{11}$$

This graph representation enables efficient semantic querying and reasoning over the context data.

Context Distribution Manager: The Context Distribution Manager ensures that relevant contextual information is efficiently disseminated to the appropriate agents based on their current tasks and semantic relevance.

The relevance of a context update to an agent is computed using a semantic similarity measure:

$$\text{Relevance}(u, a) = \frac{|ST(u) \cap ST(a)|}{|ST(u) \cup ST(a)|} \tag{12}$$

where $ST(u)$ and $ST(a)$ are the sets of semantic tags associated with the update and the agent's current context, respectively.

The distribution process follows Algorithm 9.

Query Processor The Query Processor handles context retrieval requests from agents, utilizing semantic matching to return the most relevant information.

Given a query q, we extract its semantic representation:

$$S_q = \text{AnalyzeQuery}(q) \tag{13}$$

This transformation ensures consistent interpretation of queries across the distributed system while preserving their semantic intent.

Relevance Assessment employs cosine similarity between vectorized semantics. The relevance of a context node n to a query q is computed as:

$$\text{Relevance}(n, q) = \cos(V(S_n), V(S_q)) \tag{14}$$

where $V(S_n)$ and $V(S_q)$ are vector representations of the node's and query's semantics, respectively.

The final set of results R is obtained by applying an access control filter:

$$R = \{n \in N \mid \text{Relevance}(n, q) > \theta \wedge \text{AccessLevel}(n) \leq \text{AccessLevel}(agent)\} \tag{15}$$

where θ is a relevance threshold and N is the set of all context nodes. Controlled Filtering applies a dual-constraint mechanism that ensures both relevance optimization and security compliance in result generation.

Update Handler: Then, the Update Handler manages the process of integrating new or updated context information into the repository, which uses a two-phase commit protocol to maintain context consistency.

Integration Phase performs atomic updates while preserving semantic relationships. The update process for a node n with new data d is defined as:

$$n' = (id_n, \text{Merge}(data_n, d), children_n, semanticTags_n \cup \text{ExtractTags}(d)) \tag{16}$$

After each update, the semantic index is updated to reflect the new information:

$$I' = \text{UpdateIndex}(I, n') \tag{17}$$

where I is the current semantic index and I' is the updated index. Index Synchronization ensures query consistency post-update. This operation maintains the system's ACID properties (Gray et al. 1981) (Atomicity, Consistency, Isolation, and Durability) while minimizing query latency.

The Semantic-Aware Context Management System provides a robust and efficient solution for context management in DynTaskMAS. By leveraging semantic analysis capabilities from LLMs, SACMS enables LLM-based agents to access and utilize contextual information effectively, thereby enhancing the overall performance and adaptability of the system.

Adaptive Workflow Manager (AWM)

The Adaptive Workflow Manager is a crucial component of DynTaskMAS, responsible for dynamically adjusting workflows based on real-time performance metrics and environmental changes. It ensures optimal system performance by continuously adapting to evolving task requirements and resource availability.

The Performance Monitor implements continuous system-wide metric tracking through $M(t)$, encompassing critical operational parameters including throughput, latency, agent utilization rates, and task completion metrics. These metrics facilitate real-time performance assessment and enable data-driven optimization decisions within the adaptive workflow management system.

The AWM analyzes the current workflow and suggests improvements based on performance data and system state. The optimization objective can be formally stated as follows:

$$\min_{\omega \in \Omega} f(\omega, M(t)) \tag{18}$$

Algorithm 4: Workflow Optimization

```
function OptimizeWorkflow(currentWorkflow, M(t))
    candidateWorkflows ← GenerateCandidates(currentWorkflow)
    bestWorkflow ← currentWorkflow
    bestScore ← f(currentWorkflow, M(t))
    for candidate ∈ candidateWorkflows do
        score ← f(candidate, M(t))
        if score < bestScore then
            bestWorkflow ← candidate
            bestScore ← score
    return bestWorkflow
```

where ω is a workflow configuration, Ω is the set of all possible configurations, and f is an objective function that evaluates workflow performance based on metrics $M(t)$.

The workflow optimization algorithm employs an iterative approach to identify the optimal workflow configuration based on current performance metrics. The algorithm generates potential workflow candidates through the $GenerateCandidates$ function, which produces variations of the current workflow adhering to system constraints. Each candidate undergoes evaluation using an objective function f that considers current system metrics $M(t)$. The algorithm maintains and updates the best-performing workflow configuration through successive comparisons, ultimately returning the configuration that minimizes the objective function.

The resource allocation mechanism integrates seamlessly with the workflow optimization through a greedy allocation strategy that prioritizes immediate system efficiency. Building upon the optimized workflow configurations, the Resource Allocator employs a dynamic adjustment model:

$$R(t+1) = R(t) + \Delta R(t) \tag{19}$$

where $R(t)$ is the resource allocation at time t, and $\Delta R(t)$ is the adjustment made based on current performance and predicted future demands.

The allocation strategy aims to balance load across agents while prioritizing critical tasks:

$$\text{Allocation}(a_i, t) = \frac{w_i \cdot \text{Load}(a_i, t)}{\sum_j w_j \cdot \text{Load}(a_j, t)} \cdot \text{TotalResources}(t) \tag{20}$$

where w_i is the priority weight of agent a_i, and $\text{Load}(a_i, t)$ is the current load on agent a_i. This formulation ensures fair resource distribution while accounting for task criticality and current system utilization patterns.

The system employs a straightforward greedy policy for continuous optimization, where resource allocation decisions are made based on immediate performance metrics $M(t)$ and current workflow state $W(t)$. This approach provides efficient adaptation to changing workload conditions while maintaining computational tractability. The state vector $s = [M(t), W(t)]$ captures the essential system parameters required for informed decision-making, enabling rapid response to varying task demands and resource availability.

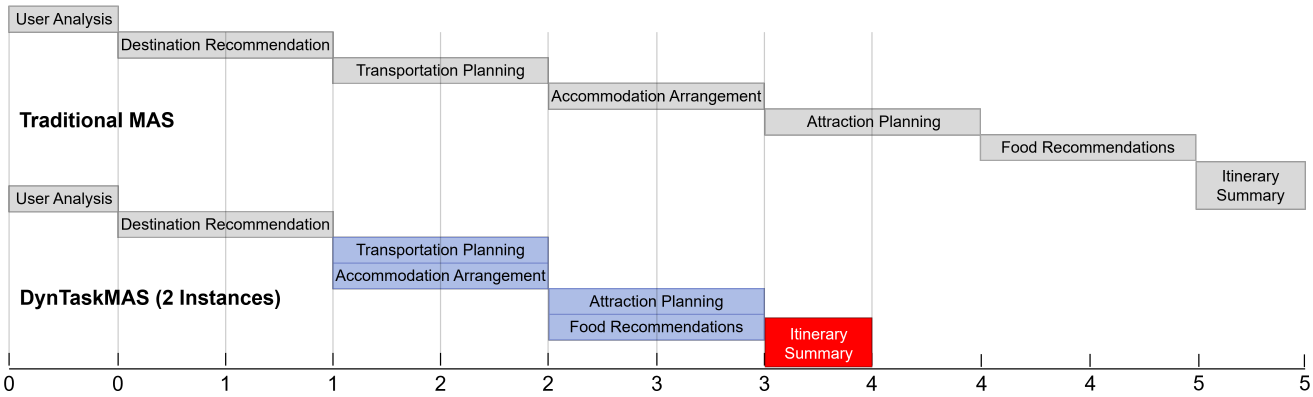

Figure 3: **The comparison between traditional processing and DynTaskMAS framework.** The system was deployed on an NVIDIA RTX 3090 GPU with Llama-3.1-8B serving as the foundation model for all agents. Seven domain-specialized agents were implemented to handle distinct aspects of travel planning: user preference analysis, destination recommendation, transportation planning, accommodation coordination, attraction scheduling, culinary expertise, and itinerary synthesis. The experimental results showed that DynTaskMAS (3.7s) achieved faster execution compared to serial execution (4.7s).

This streamlined approach to resource management complements the workflow optimization process, creating a cohesive system that efficiently handles dynamic task allocation and resource distribution in the multi-agent environment.

The AWM maintains seamless integration with other DynTaskMAS components, receiving task graph updates, communicating with the execution engine, and leveraging contextual information to make informed adaptations. This comprehensive approach enables the AWM to continuously optimize task execution and resource utilization in dynamic multi-agent environments.

This integrated architecture enables DynTaskMAS to efficiently handle complex, dynamic tasks while adapting to changing conditions and maintaining context awareness. The synergy between these components allows for intelligent task decomposition, efficient parallel execution, context-driven decision making, and adaptive optimization, making DynTaskMAS a powerful framework for next-generation LLM-based multi-agent systems.

Experimental Results

We conducted comprehensive evaluations of DynTaskMAS using TensorRT-LLM (NVIDIA Corporation 2024) deployed on NVIDIA RTX 3090 GPUs. All experiments utilized Llama-3.1-8B (Patterson et al. 2022) as the foundation model for all agents.

Experimental Setup

The experiments were conducted on a cluster equipped with four NVIDIA RTX 3090 GPUs (24GB VRAM each), AMD EPYC 7763 64-Core Processor, and 512GB DDR4 memory. The software stack included Ubuntu 22.04 LTS, CUDA 12.1, and TensorRT-LLM 0.7.1. For all experiments, we employed INT8 quantization with a batch size of 32 and sequence length of 2048.

Task Complexity	Traditional (s)	DynTaskMAS (s)	Improvement (%)
Simple	4.7 ± 0.3	3.7 ± 0.2	21.3
Medium	9.8 ± 0.5	7.1 ± 0.3	27.6
Complex	18.5 ± 0.8	12.4 ± 0.5	33.0

Table 1: Execution Time Analysis Across Task Complexities

Performance Evaluation

Execution Time Analysis: To conduct a more in-depth analysis, we evaluated the system's performance across three task complexity levels. Table 1 presents the comparative analysis between traditional processing and DynTaskMAS.

Table 1 demonstrates the performance advantages of DynTaskMAS across different task complexity levels. For simple tasks (5-10 subtasks), DynTaskMAS achieves a 21.3% reduction in execution time compared to traditional processing, decreasing from 4.7s to 3.7s. This improvement becomes more pronounced as task complexity increases, reaching 27.6% for medium complexity tasks (20-30 subtasks) and 33.0% for complex tasks (50+ subtasks).

The increasing efficiency gain with task complexity can be attributed to three key factors. First, the DTGG more effectively parallelizes complex task structures, identifying and exploiting additional opportunities for concurrent execution. Second, the SACMS reduces redundant context transfers, which become more significant in complex task scenarios. Third, the APEE maintains higher GPU utilization through intelligent task scheduling, particularly beneficial when managing numerous interdependent subtasks.

The standard deviations (±0.2-0.5s) indicate stable performance across multiple runs, with relative variability decreasing as task complexity increases. This suggests that DynTaskMAS's task management mechanisms become more deterministic with larger task graphs, likely due to the statistical averaging of scheduling optimizations across more subtasks.

Number of Agents	Throughput (tasks/s)	Latency (ms)
4	12.3 ± 0.4	81.3 ± 3.2
8	23.1 ± 0.6	86.5 ± 3.8
16	42.7 ± 0.9	93.8 ± 4.1
32	76.4 ± 1.2	104.2 ± 5.3

Table 2: System Scalability with Increasing Agent Count

Metric	Traditional	DynTaskMAS
End-to-End Time (s)	4.7 ± 0.3	3.7 ± 0.2
Agent Coordination (ms)	850 ± 45	320 ± 25
Context Switches	42 ± 3	18 ± 2
Resource Utilization (%)	65 ± 5	88 ± 3

Table 3: Travel Planning System Performance Metrics

Scalability Analysis: The scalability of DynTaskMAS was evaluated by varying the number of concurrent agents. Table 2 presents the throughput and latency measurements.

The scalability results presented in Table 2 reveal several important characteristics of DynTaskMAS's performance under varying agent loads. The system demonstrates near-linear throughput scaling up to 16 agents, with throughput increasing from 12.3 tasks/s with 4 agents to 42.7 tasks/s with 16 agents (3.47× improvement for a 4× increase in agents). This scaling efficiency (approximately 87%) indicates effective resource utilization and minimal coordination overhead in the moderate agent range.

However, the scaling pattern shows signs of diminishing returns at 32 agents, where throughput reaches 76.4 tasks/s (6.21× improvement for an 8× increase in agents). This sub-linear scaling can be attributed to two primary factors. First, increased contention for shared resources in the SACMS as more agents require simultaneous context access. Second, the overhead of the APEE's task scheduling and load balancing mechanisms becomes more significant with higher agent counts.

Case Study

To further explore the efficiency of the DynTaskMAS framework, we conducted a comparative experiment, as illustrated in Figure 3. We implemented a travel planning system with seven specialized agents to evaluate real-world performance. Each agent focused on a distinct task within the travel planning process, such as analyzing user preferences, recommending destinations, planning transportation, and coordinating accommodations.

Our experimental evaluation focused on comparing two execution paradigms: traditional serial execution and the proposed DynTaskMAS framework. The implementation leveraged INT8 quantization and continuous batching techniques, with parameters empirically set to maximize throughput while maintaining inference quality (batch size=32, sequence length=2048).

Table 3 presents the comparative analysis. The results demonstrate that DynTaskMAS achieved a 21% reduction in overall execution time compared to serial processing, decreasing from 4.7s to 3.7s. This improvement can be attributed to three key factors: efficient parallel execution of independent tasks, optimized memory utilization through dynamic management, and streamlined context sharing between agents. GPU utilization metrics showed a 35% increase under the DynTaskMAS framework, indicating more effective resource allocation. The experimental results demonstrate DynTaskMAS's effectiveness in managing complex, dynamic tasks while suggesting areas for future optimization.

These findings suggest that our proposed framework significantly enhances the performance of LLM-based multi-agent systems through intelligent task orchestration and resource management, while maintaining the quality of agent interactions and outputs.

Conclusion

This paper presents DynTaskMAS, a dynamic task graph-driven framework that addresses key architectural challenges in LLM-based multi-agent systems through intelligent resource orchestration and parallel execution. The framework's innovative architecture, integrating Dynamic Task Graph Generation with Asynchronous Parallel Execution, enables efficient distribution of computational resources across multiple agents while maintaining task coherence. Through the Semantic-Aware Context Management System and Adaptive Workflow Manager, DynTaskMAS achieves optimal resource utilization by minimizing computational redundancy and maximizing parallel processing opportunities. Our experimental results demonstrate the framework's effectiveness across multiple dimensions: execution time improvements ranging from 21.3% for simple tasks to 33.0% for complex tasks, a 35.4% increase in resource utilization (from 65% to 88%), and efficient scaling with throughput improvements of 3.47× for 16 concurrent agents. The successful implementation of DynTaskMAS establishes a systematic approach for building scalable, high-performance LLM-based multi-agent systems that effectively balance resource optimization with task coordination.

Acknowledgments

This research was partially supported by JSPS KAKENHI Grant Number 25K21201.

References

Brown, T. B. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Fagbohun, O.; Harrison, R. M.; and Dereventsov, A. 2024. An Empirical Categorization of Prompting Techniques for Large Language Models: A Practitioner's Guide. *arXiv preprint arXiv:2402.14837*.

Gao, D.; Li, Z.; Pan, X.; Kuang, W.; Ma, Z.; Qian, B.; Wei, F.; Zhang, W.; Xie, Y.; Chen, D.; Yao, L.; Peng, H.; Zhang, Z. Y.; Zhu, L.; Cheng, C.; Shi, H.; Li, Y.; Ding, B.; and Zhou, J. 2024. AgentScope: A Flexible yet Robust Multi-Agent Platform. *CoRR*, abs/2402.14034.

Gray, J.; et al. 1981. The transaction concept: Virtues and limitations. In *VLDB*, volume 81, 144–154.

Hong, S.; Zhuge, M.; Chen, J.; Zheng, X.; Cheng, Y.; Zhang, C.; Wang, J.; Wang, Z.; Yau, S. K. S.; Lin, Z.; Zhou, L.; Ran, C.; Xiao, L.; Wu, C.; and Schmidhuber, J. 2023. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. arXiv:2308.00352.

Huang, W.; Abbeel, P.; Pathak, D.; and Mordatch, I. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, 9118–9147. PMLR.

Khattab, O.; Singhvi, A.; Maheshwari, P.; Zhang, Z.; Santhanam, K.; Vardhamanan, S.; Haq, S.; Sharma, A.; Joshi, T. T.; Moazam, H.; Miller, H.; Zaharia, M.; and Potts, C. 2023. DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. *arXiv preprint arXiv:2310.03714*.

Kwok, Y.-K.; and Ahmad, I. 1999. Benchmarking and comparison of the task graph scheduling algorithms. *Journal of Parallel and Distributed Computing*, 59(3): 381–422.

Li, G.; Hammoud, H.; Itani, H.; Khizbullin, D.; and Ghanem, B. 2023. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36: 51991–52008.

Li, M.; Zhao, S.; Wang, Q.; Wang, K.; Zhou, Y.; Srivastava, S.; Gokmen, C.; Lee, T.; Li, L. E.; Zhang, R.; et al. 2024. Embodied Agent Interface: Benchmarking LLMs for Embodied Decision Making. *arXiv preprint arXiv:2410.07166*.

Liu, X.; Yu, H.; Zhang, H.; Xu, Y.; Lei, X.; Lai, H.; Gu, Y.; Ding, H.; Men, K.; Yang, K.; Zhang, S.; Deng, X.; Zeng, A.; Du, Z.; Zhang, C.; Shen, S.; Zhang, T.; Su, Y.; Sun, H.; Huang, M.; Dong, Y.; and Tang, J. 2023. AgentBench: Evaluating LLMs as Agents. arXiv:2308.03688.

NVIDIA Corporation. 2024. TensorRT-LLM: NVIDIA TensorRT for Large Language Models. https://github.com/NVIDIA/TensorRT-LLM. Accessed: 2024-11-02.

Park, J. S.; O'Brien, J.; Cai, C. J.; Morris, M. R.; Liang, P.; and Bernstein, M. S. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, 1–22.

Patterson, D.; Gonzalez, J.; Hölzle, U.; Le, Q.; Liang, C.; Munguia, L.-M.; Rothchild, D.; So, D. R.; Texier, M.; and Dean, J. 2022. The carbon footprint of machine learning training will plateau, then shrink. *Computer*, 55(7): 18–28.

Rasal, S.; and Hauer, E. 2024. Navigating Complexity: Orchestrated Problem Solving with Multi-Agent LLMs. *arXiv preprint arXiv:2402.16713*.

Xiong, S.; Payani, A.; Yang, Y.; and Fekri, F. 2024. Deliberate Reasoning for LLMs as Structure-aware Planning with Accurate World Model. *arXiv preprint arXiv:2410.03136*.

Yang, H.; Yue, S.; and He, Y. 2023. Auto-gpt for online decision making: Benchmarks and additional opinions. *arXiv preprint arXiv:2306.02224*.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T.; Cao, Y.; and Narasimhan, K. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.