IMPROVING LARGE LANGUAGE MODEL BASED MULTI-AGENT FRAMEWORK THROUGH DYNAMIC WORKFLOW UPDATE

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent frameworks powered by large language models (LLMs) have demonstrated great success in automated planning and task execution. However, the effective adjustment of workflows during execution has not been well-studied. A flexible workflow is crucial, as in many real-world scenarios, the initial plan must adjust to unforeseen challenges and changing conditions in real-time to ensure the efficient execution of complex tasks. In this paper, we define *workflows* as an *activity-on-vertex (AOV) graphs*. We continuously refine the workflow by dynamically adjusting task allocations based on historical performance and previous AOV with LLM agents. To further enhance system performance, we emphasize *modularity* in workflow design based on measuring parallelism and dependence complexity. Our proposed multi-agent framework achieved efficient sub-task concurrent execution, goal achievement, and error tolerance. Empirical results across different practical tasks demonstrate dramatic improvements in the efficiency of multi-agent frameworks through dynamic workflow updating and modularization.

1 INTRODUCTION

Large Language Models (LLMs) (Significant Gravitas; Zhou et al., 2023) show remarkable abilities to understand and generate human-like text. Recent advances have significantly enhanced their capability to emulate human reasoning (Sun et al., 2024), indicating a promising future for LLM-based reasoning. With the powerful ability to deal with a variety of natural language processing tasks, these models underpin a wide range of applications, from conversational agents (Ye et al., 2024) and content creation tools (Yao et al., 2023) to advanced analytics and decision-making systems (Ramesh et al., 2021; Wang et al., 2023). Building upon this foundation, a key advancement is the development of ***multi-agent systems empowered by LLMs*** (Liu et al., 2023; Li et al., 2023; Hong et al., 2024b; Wu et al., 2024; Wang et al., 2024; Chen et al., 2024) where multiple LLM-based agents collaborate to address the same task, leveraging their collective reasoning and planning abilities to automate and optimize task execution processes.

Existing LLMs-based multi-agent systems define LLM as an agent and agents are collaborated with each others via manually designed or LLM-generated prompts. Specifically, MetaGPT (Hong et al., 2024b) focuses on programming tasks by leveraging Standardized Operating Procedures (SOPs) (Wooldridge & Jennings, 1998; DeMarco & Lister, 2013; Belbin, 2010). It predefined distinct roles such as product manager, project manager, and engineer. For each role, an LLM agent is initialized, and these agents follow a strict and sequential workflow to execute sub-tasks. CAMEL (Li et al., 2023) is designed to complete a variety of tasks. It requires users to predefine two agents. These agents interact and execute tasks sequentially, with each agent taking on specific responsibilities. AutoGen (Wu et al., 2024) is also aimed at completing diverse tasks. Unlike CAMEL, AutoGen can automatically create an agent list with different roles based on the task requirements. These agents execute tasks sequentially following the order in the list.

Building upon the strengths of current multi-agent systems, our work aims to further improve existing general-purpose multi-agent systems by enabling ***dynamically updating workflows*** during task execution and encouraging ***modularity*** in workflows when planning the workflows.

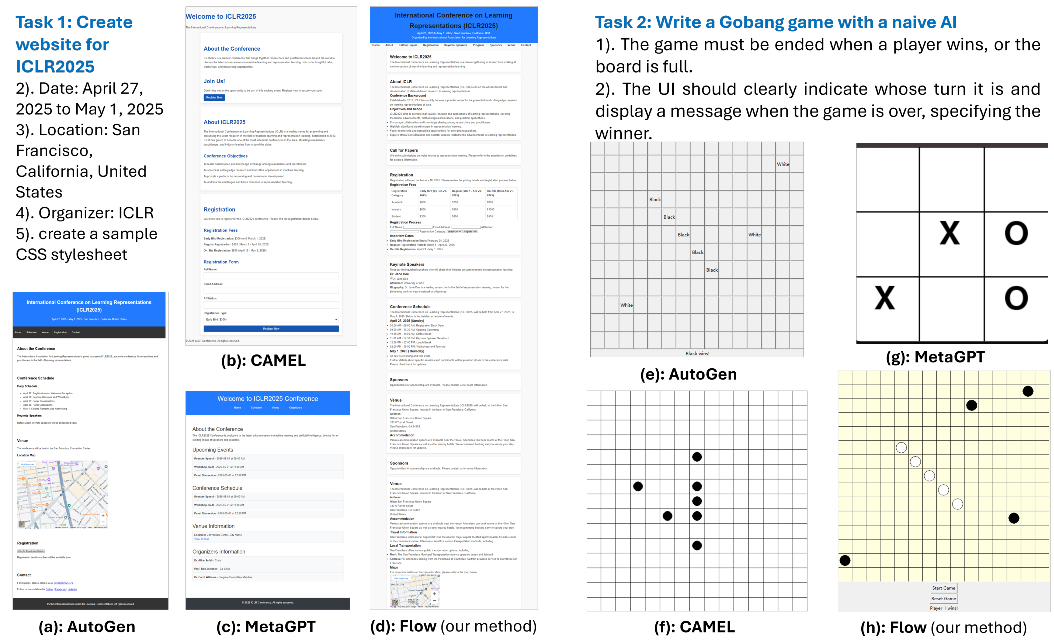


Figure 1: Comparative evaluations among four frameworks—AutoGen, CAMEL, MetaGPT, and Flow (ours)—across two tasks, present notable differences in performance. For the left task, AutoGen, CAMEL, and MetaGPT only managed to produce basic designs lacking in completeness while Flow excelled by creating a fully developed and well-structured website. For the right task, our Flow demonstrated superior capability by successfully generating a working game with a clear and intuitive interface while the other frameworks struggled to deliver fully functional or correct code.

Specifically, ***dynamic updating workflow*** allows to adjust ***sub-task allocations*** and ***agent roles*** in real-time based on ongoing performance feedback and changing conditions. This capability ***ensures*** that the system remains responsive and efficient even when faced with unexpected obstacles. For instance, if an agent encounters a roadblock in data preprocessing, dynamic updating allows the system to reassign the sub-task to another agent or introduce a new sub-task to overcome the challenge. This adaptability is essential for maintaining robustness and ensuring the seamless execution of complex tasks.

Modularization in system design involves dividing a system into separate, independently operating modules, each responsible for specific functionalities (Baldwin & Clark, 1999). A highly modularized system allows each module to be developed, managed, and executed in isolation, which simplifies system design and enhances adaptability. In our context, modularization refers to the decomposition of a complex task into smaller, interchangeable sub-task modules. A highly modularized workflow enables sub-tasks to execute concurrently, without bottlenecks from other parts of the workflow. It directly improves the operational efficiency of multi-agent frameworks. In addition, modularity dramatically enhances the ease of dynamic updating. When workflows are highly modularized, the dependency complexity between sub-tasks is small. Therefore, updating one sub-task does not necessitate changes in others, allowing for small adjustments. For instance, if an agent responsible for data preprocessing encounters an unexpected obstacle, the system can dynamically introduce a new sub-task to address the issue with little influence on the rest of the workflow.

In this paper, we have improved existing multi-agent systems by fulfilling modularity and dynamic updating workflow. Our system allows agents to run their sub-tasks in a parallel manner while enabling effective dynamic updates to workflows simultaneously by formulate the entire workflow as an Activity-on-Vertex (AOV) graph, which is a directed acyclic graph (DAG) where each sub-task is represented as a node with its status and generated logs, and the directed edges capture dependencies between sub-tasks. To encourage a modularized workflow design from the beginning, we generate multiple candidate AOV graphs for the task. These candidates are then evaluated based on their degree of parallelism and the complexity of their dependencies. The AOV graph with The highest parallelism and lowest dependency complexity is selected.

During task execution, our system dynamically updates the workflow when a sub-task fails (more detail on Fig. 2: *Running and Tracking status*). Updating the system involves modifications to task allocations and agent roles based on ongoing performance data and current workflows. As our AOV-based workflow is encouraged to have high modularity, updating one module does not necessitate changes in others, allowing for localized adjustments during workflow updates (more detail on Fig. 2: *Refining*). Similar to the initial workflow generation, multiple AOV graphs are generated and the one with the ***highest parallelism*** and ***lowest dependency complexity*** is selected during the dynamic updates. This iterative workflow refinement process ensures a good capability of adapting to new challenges and evolving objectives throughout task execution without compromising overall performance.

Our key contributions are as follows: 1) We introduce and encourage modularity in multi-agent workflows, emphasizing the design of workflows with high levels of parallelism and reduced dependency complexities. This modular design enhances efficiency, robustness, and scalability by enabling concurrent task execution and minimizing bottlenecks caused by complex interdependencies. 2) We propose a practical multi-agent framework that supports highly flexible updates to the workflow during runtime. Our method enables updates to the entire workflow based on global information, allowing agents to efficiently adapt to unexpected challenges while maintaining system coherence and consistency. 3) Through comprehensive experiments across multiple datasets, we demonstrate significant improvements in both adaptability and efficiency of our multi-agent system compared to existing approaches. The effectiveness of our method is further validated through a series of experimental evaluations.

2 Related Work

LLM-based Task Decision-Making Recent developments in LLM-driven task decision-making have focused on enhancing the reasoning and planning abilities of agents. Previous approaches like ReAct (Yao et al., 2023) which iteratively generates thoughts and actions based on current observations until task completion. This framework integrates action-taking with reasoning, allowing agents to perform complex tasks in dynamic environments. Reflexion (Shinn et al., 2023) further improves this by incorporating self-reflection, where the agent evaluates and adjusts its reasoning during execution. ADAPT (Prasad et al., 2023) introduces recursive task decomposition, enabling LLM-based agents to break tasks into smaller subtasks, leading to improved task execution flexibility. However, these approaches often overlook dynamic task reallocation, particularly in multi-agent settings, which is where our work extends the current research.

LLM-based Multi-Agent Frameworks Multi-agent frameworks have long been employed for task execution in distributed environments, with recent advancements leveraging LLMs to enhance coordination and decision-making. Current frameworks like MetaGPT (Hong et al., 2024b) and CAMEL (Li et al., 2023) use structured workflows where multiple agents collaborate to accomplish complex tasks. However, these frameworks often rely on static workflows, which limit their ability to adapt dynamically to changes in the task environment. Recent works like AutoGen (Wu et al., 2024) address this limitation by introducing more flexible agent collaboration mechanisms. Recent works have explored the use of graphs to represent workflows in multi-agent systems. DyLAN (Qian et al., 2024) and MACNET (Liu et al., 2024) utilize static workflows that remain unchanged during execution. GPTSwarm (Zhuge et al., 2024) enhances agent interactions but maintains a fixed agent topology, which may limit flexibility in task planning. DataInterpreter (Hong et al., 2024a) updates workflows primarily in response to execution failures in subtasks, adjusting subsequent tasks while leaving completed tasks unchanged. In contrast, our method encourages modularity and facilitates highly flexible modifications to the workflow during runtime, including updates to the agent topology. This capability allows our system to revise and optimize all tasks based on globally generated information, addressing both execution failures and any deficiencies in achieving the overall objectives.

3 Method

Our proposed framework enhances multi-agent frameworks powered by LLMs by introducing modularity and dynamic workflow updating. This section details how we achieve these features.

Formulating a Workflow as an AOV Graph Activity on Vertex (AOV) graph is a type of directed acyclic graph (DAG) where vertices represent tasks and edges denote precedence relations (Bondy & Murty, 2011). AOV Graphs are crucial in project scheduling and management (Moder et al., 1983; Taha, 2017), helping planners visualize dependencies and sequence tasks efficiently.

Inspired by that, we define Multi-Agent workflow as an AOV Graph where vertices represent sub-tasks, with its edges denoting dependencies between these sub-tasks. Let $G = (V, E, A)$ denote the AOV Graph, where V is the set containing all sub-tasks (nodes), $E \subseteq V \times V$ represents the set of directed edges indicating sub-task dependencies, and A represents a set of agents for all sub-tasks. Each agent $a_j \in A$ is associated with a role s_j and is responsible for executing a subset of tasks $\mathcal{T}_j \subseteq V$. We also generate sub-tasks and each directed edge $e_{ij} = (v_i, v_j) \in E$ indicates that sub-task v_i must be completed before sub-task v_j can be started.

Note that AutoGen (Wu et al., 2024) also automatically generates sub-tasks and agents. However, the sub-tasks are designed to be executed ***sequentially***. For Flow, we allow for the generation of complementary sub-tasks that can run ***in parallel***. This distinction enhances our system's ability to handle multiple tasks simultaneously, which reduces overall process time and increases efficiency.

Modularity in a Workflow Modularity in system design (Baldwin & Clark, 1999) involves dividing a system into separate, independently operating modules, each responsible for specific functionalities, allowing focus on individual components without affecting the entire system. In the context of workflows, we advocate for the creation of sub-tasks that can be executed independently. Modularity is essential for scalability and flexibility in workflows. By reducing dependency complexity, the system can more easily adapt to changes, such as the introduction of new tasks or the reassignment of existing ones, without requiring extensive restructuring.

To encourage modularity in the generated AOV Graph, we define two quantitative measures that evaluate parallelism and dependency complexity, respectively. Parallelism measures the extent to which tasks can be executed concurrently. Let S_t represent the set of tasks executed at step t. Let T be the total number of steps (the maximum depth of The DAG). Given an AOV Graph $G = (V, E, A)$, the degree of parallelism at a specific step t is defined as the average ratio of the number of tasks executed in that step to the total number of tasks:

$$P_{\text{avg}} = \frac{1}{T} \sum_{t=1}^{T} P(t), \text{ where } P(t) = \frac{|S_t|}{|V|}.$$

While P_{avg} provides a measure of parallelism, it is insufficient to fully capture the modularity that arises when sub-tasks can be executed independently. Consider two workflows, both containing the same sub-tasks $\{A, B, C, D\}$. For Workflow 1, the task dependencies are defined as: $A \to C, B \to C, A \to D, B \to D, C \to D$. In contrast, Workflow 2 has dependencies: $A \to C, B \to C, C \to D$. Although both workflows exhibit the same level of parallelism, Workflow 2 is structurally simpler in terms of task dependencies, as it contains fewer edges.

To account for this complexity, we measure the dependency structure by analyzing the degree distribution within the task graph. For each task v_i, the degree $\deg(v_i)$ reflects the number of direct connections it has in the graph G. The average degree $\bar{d}$ is computed as:

$$\bar{d} = \frac{1}{|V|} \sum_{v_i \in V} \deg(v_i),$$

where $|V|$ is the number of tasks (vertices) in the graph. The complexity of task dependencies is then quantified by the standard deviation of the degree distribution:

$$C_{\text{dependency}} = \left(\frac{1}{|V|} \sum_{v_i \in V} (\deg(v_i) - \bar{d})^2 \right)^{\frac{1}{2}}.$$

This measure reflects the variability in the number of dependencies each task has, providing insight into the overall complexity of the workflow structure.

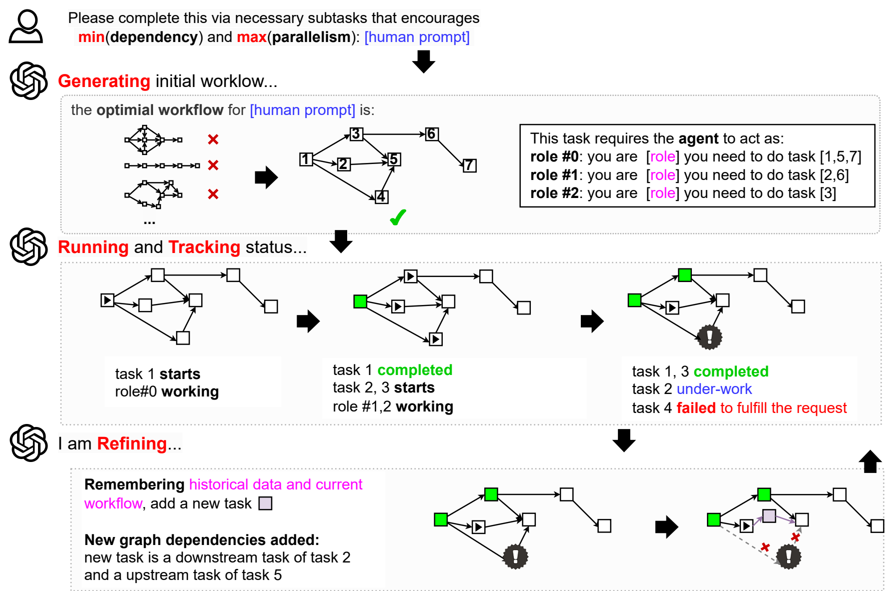


Figure 2: Our system encourages the modularity of the sub-tasks and allows agents to run their tasks in a parallel manner while enabling dynamic updates to workflows simultaneously.

Task dependencies alone are insufficient to fully capture the modularity that allows sub-tasks to be executed independently. Consider Workflow 3: $A \rightarrow B \rightarrow C \rightarrow D$, which may have a similar dependency complexity to Workflow 2. However, Workflow 2 provides greater modularity and separation of tasks, highlighting the importance of evaluating both dependency complexity and modularity to fully assess and promote effective workflow designs. Both measures are essential for ensuring that tasks can be executed in parallel while maintaining a well-structured, modular approach.

Prompt for Initialization P_{init}

```
You are an intelligent workflow planner. Given the following task requirements,
     generate a set of necessary sub-tasks along with their dependencies and assign
     appropriate agents to each task. Ensure that tasks that can be executed in
     parallel are identified to enhance efficiency. The workflow should be represented
     as a dictionary where each key is a task and its value contains the task's status,
      data, number of parents not completed, child tasks, and assigned agent.

Task Requirements: {TASK_REQUIREMENTS}

Output Format: { "Task_A": { "status": "not started", "data": null, "
     num_parents_not_completed": 0, "child": ["Task_B", "Task_C"], "agent": "Agent_1"
     }, "Task_B": { "status": "not started", "data": null, "num_parents_not_completed":
      1, "child": ["Task_D"], "agent": "Agent_2" }, ... }
```

Generate an Initial AOV Graph Given a task requirment T, firstly, we prompt a LLM f to generate a set of candidate AOV Graphs $\{G_1, G_2, \dots, G_K\}$ based on the task requirements and our ***Prompt for initialization*** P_{init}, i.e., $\{G_1, G_2, \dots, G_K\} = f(P_{\text{init}}, T)$. Each candidate AOV Graph $G_k = (V_k, E_k, A_k)$ is evaluated using the measures of parallelism and dependency complexity. We prioritize the workflow with the highest parallelism score. If after the selection, the graph is not unique, we further select the one with the lowest dependency complexity.

Note that we prioritize parallelism and modularity early in the process and focus on refining the workflow through data-driven adjustments during runtime. The reasons are as follows: 1) When leveraging LLMs to generate workflows for specific tasks, these models inherently possess reasoning capabilities that make the workflows reasonably reliable, even without explicitly emphasizing reliability in the prompts. However, the specific task can often be achieved through multiple workflows, many of which may not prioritize efficiency. If parallelism and independence are not explicitly encouraged during the initial workflow generation, the model might produce sequential or overly complex workflows, making them inefficient. Therefore, we emphasize parallelism and modularity from the outset. 2) We do not have additional data to verify correctness, and without such data, verifying correctness becomes inherently challenging. This is similar to the scientific process, where experimental validation and iterative refinements are necessary to improve the accuracy of physical laws. Since no supervised information is available at the beginning, we focus on refining the workflow during runtime as data becomes available.

Execution Plan Generation and Agent Allocation After we get the best candidate for the AOV Graph, We begin by performing a topological sort on the task dependency graph to determine the order of task execution. The topological sort produces a linear ordering of the tasks $\sigma : V \to \{1, 2, \ldots, |V|\}$ such that for any edge $(v_i, v_j) \in E$, $\sigma(v_i) < \sigma(v_j)$. The result is a sequence of task steps, where each step consists of tasks that can be executed in parallel. This execution plan minimizes the number of steps needed to perform while ensuring that all tasks are completed in the shortest possible time, adhering to their dependencies.

Each agent $a_j \in A$ is associated with a set of sub-tasks $\mathcal{T}_j \subseteq V$, indicating the tasks that the agent is capable of handling. In our framework, we allow for the reuse of agents across different tasks based on their roles and time availability. However, if two sub-tasks v_p and v_q require the same agent a_j at the same step s_i, we create a clone of the agent, denoted a'_j, to run both sub-tasks simultaneously without increasing the wall time.

Prompt for Update P_{update}

```
You are an intelligent workflow updater. Based on the current workflow and the all
     subtasks' progress data, update the workflow for acheving the objective by adding,
      removing, or modifying subtasks as necessary. Ensure that the updated workflow
     maintains modularity and maximizes parallel execution.
Output Format: { "Task_A": { "status": "not started", "data": null,  ... }
```

Workflow Refinement and Dynamic Updating Our dynamic updates are designed to be flexible, allowing modifications to task allocations including deletion, addition, editing, rerunning, and reassignment of agents. Without a modularity constraint, such flexibility would be difficult to implement. For instance, subtask dependencies can be very complex, and dynamically changing a task could necessitate redoing many existing tasks or incorporating many new tasks. With modularity, efficiency in our dynamic updating process is dramatically enhanced. Intuitively, when workflows are modular, updating one module does not necessitate changes in others.

We leverage Large Language Models (LLMs) as a global inspector to update an AOV Graph based on global information. Specifically, given task requirements T, a prompt for update P_{update}, the current AOV Graph G^t, and generated data D^t containing the status of subtasks and the output of agents for running subtasks, the LLM continuously monitors task progress and dynamically modifies the graph when necessary. Similar to the initialization process, we also generate K candidate graphs: $\{G_1^{t+1}, G_2^{t+1}, \ldots, G_K^{t+1}\} = f(P_{\text{update}}, T, D^t)$. We follow the same selection strategy as in initialization which prioritizes the workflow with the highest parallelism score. If after the selection, the graph is not unique, we further select the one with the lowest dependency complexity.

Note that with sufficient data and computational resources, we could further enhance our framework by fine-tuning LLMs with reinforcement learning (RL) for workflow generation. For example, the LLM would be trained to maximize a reward function designed around key performance indicators such as task completion speed, resource utilization, and minimization of workflow disruptions.

Implementation Our framework employs a dictionary-based structure, $\tilde{G}$, to efficiently manage and dynamically update workflows within a multi-agent system. This approach represents each task v in the workflow as a key in $\tilde{G}$, with the value being another dictionary that encapsulates various attributes of the task. The structure is specifically defined as:

$\tilde{G}[v] = \{$"sub-task requirement", "status", "data", "num_parents_not_completed", "child", "agent"$\}$.

Each task's dictionary includes attributes such as the sub-task requirement, current status (e.g., "not started", "in progress", "completed"), data relevant to the task, a count of uncompleted parent tasks to manage dependencies, a list of child tasks that depend on the current task's completion, and the agent assigned to the task. The choice of a dictionary-based structure for our workflow system is driven by its inherent simplicity and flexibility. This structure can be converted directly to JSON, and the organized information is easily readable and summarizable by large language models (LLMs).

Each task's execution readiness is determined by the attribute "num_parents_not_completed". Tasks with a count of zero are eligible to run concurrently, leveraging our system's capability to handle parallel task execution effectively. Upon the completion of any task, we perform a systematic review to determine if the workflow requires refinement, ensuring that all dependencies are accurately accounted for and that the workflow remains aligned with project goals. Additionally, we do not rely solely on the status and "num_parents_not_completed" counts reported by agents. These are always double-checked to prevent errors that could arise from inaccurate reporting by agents or unforeseen system anomalies. This rigorous verification process enhances the reliability and integrity of our workflow management system.

4 EXPERIMENTS

Baselines In all experiments, we compare Flowto the exists multi-agent frameworks *i.e.* (1) AutoGen (Wu et al., 2024), (2) Camel (Li et al., 2023), and (3) MetaGPT (Hong et al., 2024b). In our experiments, we use agents empowered by GPT-4o-mini and GPT-3.5 (OpenAI, 2024).

Experiment Design We designed three diverse and engaging tasks to evaluate multi-agent collaboration frameworks: 1) website development, 2) LaTeX Beamer slide creation, and 3) interactive game development. The rationale behind selecting coding-based experiments is twofold. First, most multi-agent frameworks tend to favour coding and writing abilities, like MetaGPT (Hong et al., 2024b). Using non-coding tasks may introduce bias. Second, coding tasks effectively demonstrate the system's ability to assign agents and manage task allocation. *Development of a Gobang Game with Naive AI*: This task requires creating a Gobang (Five in a Row) game with a user interface and a simple AI opponent. Players can choose between black or white stones, with the UI clearly indicating turns and announcing the winner or draw when the game ends. This task demonstrates the system's ability to handle modular design and task parallelism, as it involves coordinating game logic, AI implementation, and user interface development simultaneously.

Machine Learning Course Lecture Slides: This task focuses on generating LaTeX slides covering reinforcement learning algorithms, including motivations, problem statements, intuitive solutions, and detailed mathematical equations. A specific page requirement is to test the system's ability to follow instructions precisely. The task highlights the system's parallel processing capabilities of simultaneous generation of content, formatting, and presentation structure. The structured format of LaTeX also tests how effectively the system manages modularity and concurrent tasks.

Development of a Comprehensive Website for ICLR 2025: This task involves building a professional website for the International Conference on Learning Representations, hypothetically scheduled for San Francisco from April 27 to May 1, 2025. The website must feature key elements such as a detailed conference schedule and venue information with an interactive map. This task assesses each system's ability to manage parallel workflows and modular components, including user interface design, functionality, and adherence to design guidelines, showcasing how well the system handles task decomposition and execution.

4.1 EVALUATIONS OVER THREE DESIGNED TASKS

Evaluation Metrics To conduct both quantitative and qualitative evaluations, we employed two metrics: ***Success Rate*** and ***Human Rating***. ***Success Rate***: The Success Rate is a quantitative measure that ranges from 0 to 1. Assesses whether the multi-agent system successfully generates executable outputs that fully meet the task requirements. A higher score indicates a greater level of success in accurately fulfilling the task objectives. ***Different tasks may have different evaluation metrics. The description for each evaluation metric is defined in Appendix D.3, B.2 and B.3.***

Human Rating: Human ratings are used to evaluate the quality of the generated results in alignment with the task description. We gathered 50 participants with programming and machine learning backgrounds to rank the outcomes produced by different methods. the detailed description of how we take scores is shown in the Appendix A

Summary We here give a summary of the performance of different methods over three tasks from Table 1, 2 and 3, comparing the overall score regarding the success rate and human rating. The overall score of Flow and human rating over three tasks, are (100, 4) on game design, (100, 3.33) on LaTeX writing, and (80, 3.28) on website design. Therefore, the average performance of Flow is 93% success rate and 3.54 over 4 satisfaction. Similarly, we have the average performance of AutoGen as (66.7, 2.75), MetaGPT as (71, 1.60), and CAMEL as (48.67, 2.12). Overall, our method Flow has finished tasks with the most satisfaction and the highest success rate. Information about Flow's workflow on those task is in Appendix D

4.2 Result for Gobang Game

The experimental setup is thoroughly detailed in Appendix B.2 and the visualisation result is in Fig.1. As shown in Table 1, our method gets 100 for all the aspects regarding success rate as well as the highest satisfaction from humans. More explanations for each method are as follows:

AutoGen: With the five tests, one trail failed to generate a valid output. Of the four successful attempts, one contained a code error that hindered normal execution, while another exhibited a bug in the game interface. The remaining two tests were completed successfully, though the chess pieces were displayed as the text "black" and "white".

MetaGPT: After running MetaGPT five times, all attempts were successful and intractable. However, in four cases, a Tic-Tac-Toe game was generated instead of Gobang; out of these, the left one were functional, allowing both the user and AI to make moves and correctly terminate.

CAMEL: In all five trials, CAMEL was only successful twice. In the other attempts, the generated Python code was not executable. In the two successful trials, CAMEL successfully implemented correct termination conditions but had no AI component and terminated message.

Flow: After five rounds of testing, our system consistently generated successful outputs without any errors. The game functioned as expected, allowing both the player and the naive AI to take turns seamlessly. The game also ended correctly when either the board was fully occupied or one side achieved victory. In the game interface, the chess pieces were represented by actual black and white pieces, rather than text labels.

4.3 Result for LaTeX Beamer Writing

Experimental results are presented in Table 2 with explanations as follows:

AutoGen: After five tests, AutoGen successfully generated outputs every time. However, one output failed to compile in LaTeX due to syntax errors, and in two instances, the outputs did not meet the required length. The remaining outputs met both the length and content requirements.

MetaGPT: In five trials, four of them successfully generated a valid LaTeX version, with the only error being related to writing Python code within the '.tex' file. In these four successful trials, all documents met the required content specifications, but the total page count fell short of the requirements of 30 pages or 20 pages.

CAMEL: Successfully generated five different '.tex' files that are valid and could be rendered into Beamer format. Each presentation contained the required information, including sections like motivation. However, none of them met the page count requirement of 30 pages or 20 pages.

Flow: After five tests, our system successfully generated outputs each time, and all outputs were able to compile in LaTeX. However, one output contained repetitive content. In the remaining valid

Table 1: Comparison of different LLM-based Multi-Agent frameworks on Gobang Game

Model	Success Rate (%)				Human Rating
	Compilable	Intractable	Game Rule	Overall Score	(1-4)
AutoGen (Wu et al., 2024)	80	60	40	60	2.26
MetaGPT (Hong et al., 2024b)	**100**	**100**	20	73	1.24
CAMEL (Li et al., 2023)	40	40	0	27	2.50
Flow (Ours)	**100**	**100**	**100**	**100**	**4.00**

Table 2: Comparison of different LLM-based Multi-Agent frameworks on LaTeX Beamer writing

Model	Success Rate (%)				Human Rating
	Compilable	Completeness	Page Limit	Overall Score	(1-4)
AutoGen (Wu et al., 2024)	80	80	40	67	3.00
MetaGPT (Hong et al., 2024b)	80	80	20	60	1.83
CAMEL (Li et al., 2023)	**100**	**100**	0	66	1.83
Flow (Ours)	**100**	**100**	**100**	**100**	**3.33**

Table 3: Comparison of different LLM-based multi-agent frameworks on Website Design.

Model	Success Rate (%)				Human Rating
	Compilable	Basic Information	Sections	Overall Score	(1-4)
AutoGen (Wu et al., 2024)	80	80	60	73	2.62
MetaGPT (Hong et al., 2024b)	**100**	**100**	40	**80**	1.72
CAMEL (Li et al., 2023)	80	80	0	53	2.02
Flow (Ours)	80	80	**80**	**80**	**3.28**

outputs, the length of the Beamer presentations met the specified requirements, and all the content mentioned in the requirements was adequately covered.

4.4 Result For Website Design

Similar to the previous two, the detailed experiment set-up is in Appendix B.3. We here illustrate the results in Table 3 as follows:

AutoGen: AutoGen produced HTML and CSS files with key information displayed but lacks details. Each section of the website contains only one or two sentences, lacking interactive functionality and necessary elements such as maps or tables.

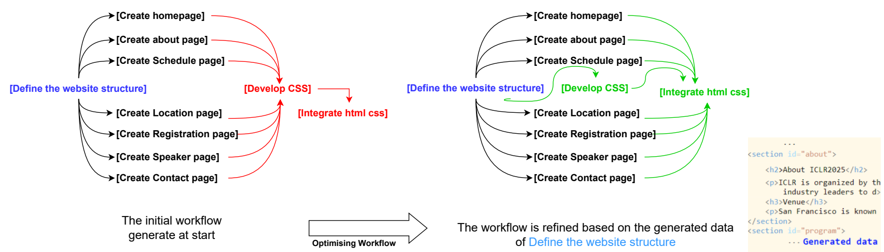
MetaGPT: MetaGPT managed to create complete HTML and CSS, meeting basic functionality requirements and showcasing its code generation capabilities. However, the outputs were overly simplistic, missing significant content and key functional modules like the required venue and map.

CAMEL: CAMEL's Outputs were executable in four out of five runs, though they did not include all the necessary elements, achieving all basic functions only. The system limits the communication can be only between two agents regardless of task complexity hindering its ability to fully complete complex website development tasks that require multi-task collaboration. Notably, one run generated complete HTML code but omitted the CSS file, preventing proper rendering of the website.

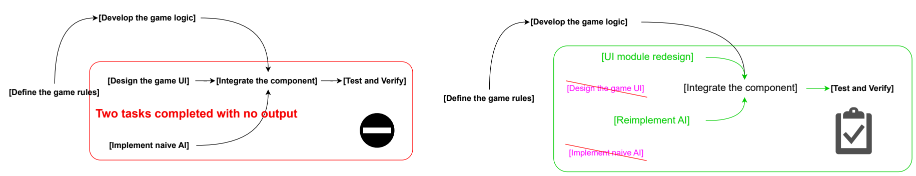
Flow: Flow achieves an 80% success rate within 5 trials. One trial failed to generate an HTML website. Among the remaining four trials, each section of the website featured detailed introductions and necessary interactive functionalities. For example, the venue section included travel information and local transportation options like airport, and accurately presented the conference location on a map. The registration section was fully functional, with a complete table, input boxes, and a submission button.

5 Workflow Update

Update based On Generated Data Fig. 3(a) demonstrates the update process of Flow in the conference website creation example. Upon completing the first subtask, the system identifies potential changes and redundancies, triggering a restructuring process to enhance efficiency. Once the task "Define the website structure" is completed, the generated data, which includes HTML structures and elements is sufficient to proceed with the CSS creation. As a result, the workflow is updated to incorporate the development of CSS based on the completed "Define the website structure" task.



(a) Conference website: no newly added subtask, only the workflow is updated.



(b) Gobang Game: bad subtasks exist, add two new subtasks for successfully completing this task.

Figure 3: Workflow and dynamic update in two cases.

Fig. 3(b) illustrates a result of our dynamic updating process, where the system, upon receiving information about completed tasks, decides to add a bridging task to handle gaps and ensure the workflow continues smoothly.

Table 4: Success Rate (%) of Error handling with dynamically updating.

Task	**Flow w/o Update**	**Flow**
Website Design	46	**87**
Gobang Game	0	**93**
LaTeX Beamer Writing	67	**93**

Error handling To evaluate the effectiveness of our updating mechanism, we intentionally introduced random masking to certain task outputs, replacing them with "none" before passing them to the next agent. We conducted five trials and recorded the success scores. Since other frameworks employ a sequential workflow, we limit the comparison to our own approach in this context.

We observed a significant difference in success rate between using dynamic updating and not, particularly in the Interactive Game section as shown in Table 4. The main issue arises when the previous agent fails to provide the necessary information, yet the second agent continues with its task, leading to a major disconnect in the code. This often results in Python being unable to compile due to missing or mismatched components. Similarly, in website design, the lack of required elements caused by this failure impacts the overall functionality and structure. During the execution of subtasks, errors may arise due to the limitations of the LLM-based agent or underperformance in certain tasks. Therefore, the ability of a multi-agent system to address such issues is essential.

6 Conclusion

We present Flow, a novel LLM-empowered multi-agent system that can dynamically adapt to unforeseen challenges for general tasks executions. With dynamically update the workflow by AOV graphs, our system has largely fulfilling the modularity requirements for completing complex tasks. We demonstrate our method through case studies on a series of experiments, ranging from website design, game development and LaTeX Beamer creation as well as testing its capability on solving general benchmark tasks. Through objective evaluation metric and human feedback, we found Flow is to able to continuously enhance the flexibility during agent collaboration and thus significantly improve the execution efficiency with improved error tolerance and better performance.

REFERENCES

Carliss Y. Baldwin and Kim B. Clark. *Design Rules: The Power of Modularity Volume 1*. MIT Press, Cambridge, MA, USA, 1999. ISBN 0262024667.

R.M. Belbin. *Team Roles at Work*. Butterworth-Heinemann, 2010. ISBN 9781856178006. URL https://books.google.com.au/books?id=hF2yJzYfUBAC.

A. Bondy and U.S.R. Murty. *Graph Theory*. Graduate Texts in Mathematics. Springer London, 2011. ISBN 9781846289699. URL https://books.google.com.au/books?id=HuDFMwZOwcsC.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=EHg5GDnyq1.

T. DeMarco and T.R. Lister. *Peopleware: Productive Projects and Teams*. Addison-Wesley, 2013. ISBN 9780321934116. URL https://books.google.com.au/books?id=DVlsAQAAQBAJ.

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Chenxing Wei, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Xiangru Tang, Xiangtao Lu, Xiawu Zheng, Xinbing Liang, Yaying Fei, Yuheng Cheng, Zhibin Gou, Zongze Xu, and Chenglin Wu. Data interpreter: An llm agent for data science, 2024a. URL https://arxiv.org/abs/2402.18679.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=VtmBAGCN7o.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Zhiwei Liu, Weiran Yao, Jianguo Zhang, Le Xue, Shelby Heinecke, Rithesh Murthy, Yihao Feng, Zeyuan Chen, Juan Carlos Niebles, Devansh Arpit, Ran Xu, Phil Mui, Huan Wang, Caiming Xiong, and Silvio Savarese. Bolaa: Benchmarking and orchestrating llm-augmented autonomous agents, 2023. URL https://arxiv.org/abs/2308.05960.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic llm-powered agent network for task-oriented agent collaboration, 2024. URL https://arxiv.org/abs/2310.02170.

J.J. Moder, C.R. Phillips, and E.W. Davis. *Project Management with CPM, PERT, and Precedence Diagramming*. Van Nostrand Reinhold, 1983. ISBN 9780442254155. URL https://books.google.com.au/books?id=WmhRAAAAMAAJ.

OpenAI. Gpt-4o mini: Advancing cost-efficient intelligence. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/, 2024. Accessed: 2024-09-29.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. Adapt: As-needed decomposition and planning with language models. *arXiv*, 2023.

Chen Qian, Zihao Xie, Yifei Wang, Wei Liu, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Scaling large-language-model-based multi-agent collaboration, 2024. URL https://arxiv.org/abs/2406.07155.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation, 2021. URL https://arxiv.org/abs/2102.12092.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vAElhFcKW6.

Significant Gravitas. AutoGPT. https://github.com/Significant-Gravitas/AutoGPT. MIT License.

Hongda Sun, Weikai Xu, Wei Liu, Jian Luan, Bin Wang, Shuo Shang, Ji-Rong Wen, and Rui Yan. DetermLR: Augmenting LLM-based logical reasoning from indeterminacy to determinacy. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9828–9862, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.531. URL https://aclanthology.org/2024.acl-long.531.

H.A. Taha. *Operations Research an Introduction*. Pearson, 2017. ISBN 9780134444017. URL https://books.google.com.au/books?id=HbpKjwEACAAJ.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023. URL https://arxiv.org/abs/2305.16291.

Yaoxiang Wang, Zhiyong Wu, Junfeng Yao, and Jinsong Su. Tdag: A multi-agent framework based on dynamic task decomposition and agent generation, 2024. URL https://arxiv.org/abs/2402.10178.

Michael Wooldridge and Nicholas R. Jennings. Pitfalls of agent-oriented development. In *Proceedings of the Second International Conference on Autonomous Agents*, AGENTS '98, pp. 385–391, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 0897919831. doi: 10.1145/280765.280867. URL https://doi.org/10.1145/280765.280867.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Yining Ye, Xin Cong, Shizuo Tian, Yujia Qin, Chong Liu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. Rational decision-making agent with internalized utility judgment, 2024. URL https://openreview.net/forum?id=l1pNNQSzZv.

Wangchunshu Zhou, Yuchen Eleanor Jiang, Long Li, Jialong Wu, Tiannan Wang, Shi Qiu, Jintian Zhang, Jing Chen, Ruipu Wu, Shuai Wang, Shiding Zhu, Jiyu Chen, Wentao Zhang, Xiangru Tang, Ningyu Zhang, Huajun Chen, Peng Cui, and Mrinmaya Sachan. Agents: An open-source framework for autonomous language agents. 2023. URL https://arxiv.org/abs/2309.07870.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs, 2024. URL https://arxiv.org/abs/2402.16823.

APPENDIX

CONTENTS

A Human Evaluation Process

Sometimes, LLMs can correctly fulfill each requirement of a task, but the quality of completion may vary. In such cases, human evaluation is necessary to assess the quality of the output. For each task, the final output of each Multi-Agent framework was evaluated by 50 participants, who ranked the outputs from best to worst. Points were awarded based on the rankings, with 1st place receiving 4 points and 2nd place receiving 3 points and so on. The final result was determined by calculating the average score. The detail distribution is shown in Fig. 5

B Experiment setups

B.1 Experiment setup: LaTeX Beamer Writing

User input

```
I am a lecturer teaching a machine learning course to research students, I am
     preparing lecture slides on various reinforcement learning algorithms.
Note that:
1). Given that the lecture duration is 2 hours, the slides should span approximately
     30 pages.
2). For each reinforcement learning algorithm covered, the slides will include the
     following key components: the motivation behind the algorithm, the problem it aims
      to solve, an intuitive solution, and the detailed mathematical equations that
     underpin the method.
3).  It is essential that the lecture is comprehensive and self-contained, providing
     students with a clear understanding of each algorithm from both a conceptual and
     technical perspective.
```

The task involves generating a LaTeX Beamer presentation, which is a popular LaTeX class used for creating professional-quality slides with various templates and effects. In this experiment, the objective is to produce presentations with different configurations, assessing the system's ability to follow instructions. The experiment includes the following configurations:

Config 1: A 30-slide presentation, including motivation, problem statement, intuitive solution, and detailed mathematical equations.

Config 2: A 20-slide presentation, including motivation, problem statement, intuitive solution, and detailed mathematical equations.

Config 3: A 30-slide presentation, including motivation, problem statement, intuitive solution, and pseudocode.

Config 4: A 20-slide presentation, including only motivation and intuitive solution.

Config 5: A 30-slide presentation, including motivation, problem statement, intuitive solution, and detailed mathematical equations.s

The goal is to examine the system's ability to follow specific instructions while generating over 20 and 30 slides in different scenarios.

This task is well-suited for evaluation because it requires not only text generation but also an understanding of formatting and presentation logic. It serves as a comprehensive test of multitasking and reasoning capabilities. The structured nature of LaTeX allows for a rigorous assessment of the agent's ability to manage complex, multi-component tasks, thereby highlighting the strengths of our method.

Evaluation Metrics: The following metrics are used to assess the performance of the generated LaTeX Beamer presentations:

(1) **Compilable:** Verifies whether the LaTeX code compiles into a valid Beamer presentation. A successful compilation is rewarded with a score of 1, otherwise 0.

(2) **Completeness:** Ensures that the final Beamer presentation includes all required components: motivation, problem, intuitive solution, and equations. Missing any of these results in a score of 0.

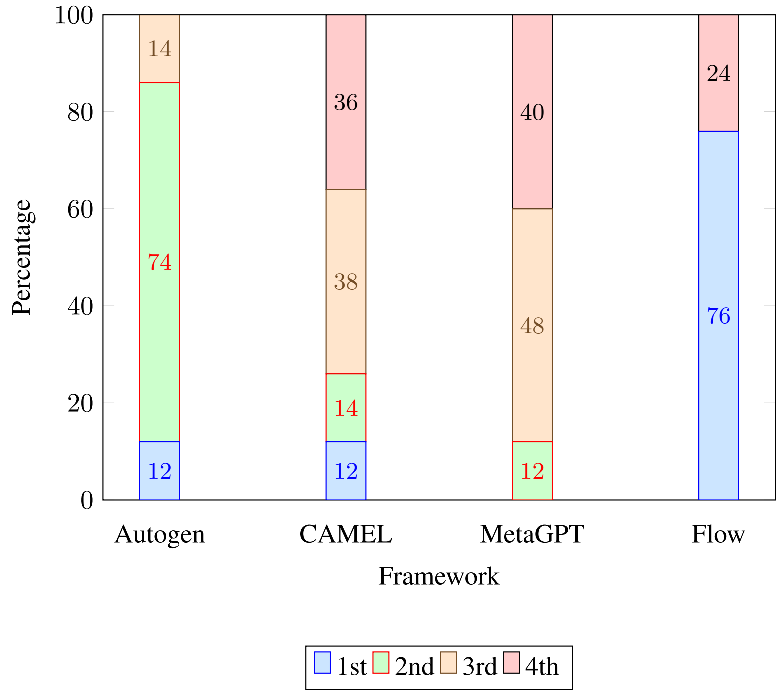


Figure 4: Ranking Distribution for conference website design. Shows that our results are better in the task of conference website design.

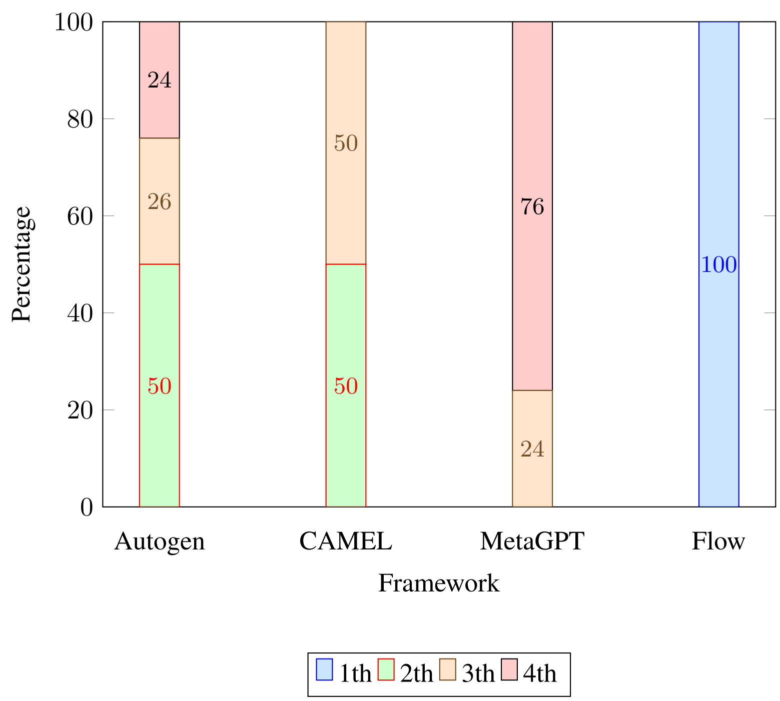


Figure 5: Ranking Distribution for Gobang game making. Shows that our results are better in the task of Gobang game making.

(3) **Page Limit:** Assesses whether the presentation adheres to the specified page limits as outlined in the prompt.

The final result is calculated as the average of these three scores and shown in percentage.

B.2 Experiment setup: Go-bang game

User input

```
I am developing a Gobang game that includes a naive AI and a user interface. The game
     should end when either a player wins or the board is completely filled. The user
     interface must clearly indicate whose turn it is and display a message when the
     game concludes, specifying the winner. Additionally, the user should have the
     option to play as either black or white stones.
```

Gobang, also called Five in a Row, is a strategy board game where two players take turns placing black and white pieces on a grid. The objective is to be the first to align five consecutive pieces in a horizontal, vertical, or diagonal line. This experiment assesses our system's ability to efficiently develop the game by utilizing parallelism to divide the development process into smaller, manageable tasks, such as game logic, AI move generation, and user interface (UI) design. We apply the same approach, taking the average score from five trials.

Evaluation Metrics: The following metrics are used to assess the performance of the generated Gobang game:

(1) **Compilable:** The code compiles without errors. Any error that causes a termination will result in a score of 0.

(2) **Interactable:** Properly supports both user and AI moves. If both functions are achieved score 1 else 0

(3) **Game Rule:** Ends correctly when five pieces are aligned, correct terminated will result in 1 final score.

Each of these metrics is scored as 0 or 1, and the final result is calculated as the average of these scores and turn into percentage. These metrics allow for a comprehensive assessment of the efficiency, accuracy, and adaptability of each framework in developing a functional Gobang game with AI capabilities.

B.3 Experiment setup: Website Design

User input

```
I am designing a website for the International Conference on Learning Representations
     (ICLR2025), which will take place from April 27, 2025, to May 1, 2025, in San
     Francisco, California, United States. The conference is organized by the
     International Association for Learning Representations.
Note that:
1). For each section, I would like to see example HTML content. Additionally, a sample
      CSS stylesheet should be provided to style the website. The content must be
     professional, clear, and appropriate for an international academic conference.
2). The website should include all the provided details, including a comprehensive
     conference schedule and a section dedicated to the conference venue, featuring a
     map.
```

We tasked the systems with developing a comprehensive website for the ICLR conference to evaluate their ability to handle complex tasks that require both flexible task coordination and effective problem-solving. This task tested the systems' ability to manage multiple interdependent steps, such as designing user interfaces, ensuring functionality, and adhering to specific design guidelines.

Evaluation Metrics: The following metrics are used to assess the performance of the generated website:

(1) **Compilable:** Checks if the HTML renders into a functioning website, If yes then score 1, can't render will result of score 0

(2) **Basic Information:** Verifies the presence of essential details like conference name, date, location, and organizer. Missing any of the information will caused the score to be 0

(3) **Sections:** Ensures inclusion of all required sections, with a focus on the schedule and venue as prompt asked. Missing required part in prompt will result of 0 in score.

By presenting a real-world scenario involving intricate requirements, we were able to observe how well the systems could break down a large project into manageable components and coordinate efforts across different tasks.

B.4 How Different LLMs Affect Updates

To verify how our framework performs with different capabilities of LLMs, we tested it on three tasks we designed using both GPT-4o-mini and GPT-3.5-Turbo. In this experiment, each task was run five times on different models, and the average of the results was calculated as the final outcome. We recorded three metrics: average init task, average changed task, and average changed ratio.
Init task refers to the number of tasks that need to be executed within the workflow after selecting the optimal workflow but before execution begins.
Average changed task indicates the number of tasks in the original workflow that were updated after completing the workflow execution.
Average changed ratio is calculated by dividing the average changed task by the init task, providing a more intuitive reflection of the proportion of tasks that were updated.

Table 5: Update information on GPT-3.5-Turbo and GPT-4o-mini

LLM-Agent	Task	Initial Tasks (avg.)	Changed Tasks (avg.)	Changed Ratio (avg.)
GPT-3.5-Turbo	Gobang Game	7.8	3.4	44%
	Website Design	7.2	4.8	66%
	LaTeX Beamer Writing	6.2	4.4	71%
GPT-4o-mini	Gobang Game	8	2.8	35%
	Website Design	7.2	3.4	47%
	LaTeX Beamer Writing	9.2	4.8	53%

Additionally, we have included a lower-performance model, GPT-3.5-Turbo, in our evaluation. As expected, GPT-3.5-Turbo required more updates during runtime as expected. This is because GPT-3.5-Turbo has comparatively weaker task execution capabilities, resulting in more frequent workflow adjustments due to insufficiently generated data.

B.5 How Different LLMs Affect Performance

In this experiment, we utilized the GPT-3.5-Turbo model to conduct experiments on three tasks across different frameworks. Each task was executed five times. We evaluated the results using the same scoring matrix described above.

Table 6: Comparison of LLM-based Multi-Agent frameworks on Gobang Game with GPT-3.5-Turbo

Model	Success Rate (%)			
	Compilable	Intractable	Game Rule	Overall Score
AutoGen (Wu et al., 2024)	80	20	20	40
MetaGPT (Hong et al., 2024b)	80	20	40	53
CAMEL (Li et al., 2023)	80	80	40	67
Flow (Ours)	**100**	**100**	**60**	**87**

Table 7: Comparison of LLM-based Multi-Agent frameworks on Website Design with GPT-3.5-Turbo

Model	Success Rate (%)			
	Compilable	Basic Information	Sections	Overall Score
AutoGen (Wu et al., 2024)	20	0	0	7
MetaGPT (Hong et al., 2024b)	80	60	**60**	67
CAMEL (Li et al., 2023)	40	40	20	33
Flow (Ours)	**100**	**100**	40	**80**

Table 8: Comparison of LLM-based Multi-Agent frameworks on LaTeX Beamer Writing with GPT-3.5-Turbo

Model	Success Rate (%)			
	Compilable	Completeness	Page Limit	Overall Score
AutoGen (Wu et al., 2024)	40	0	0	13
MetaGPT (Hong et al., 2024b)	20	20	0	13
CAMEL (Li et al., 2023)	80	80	0	53
Flow (Ours)	**100**	**100**	0	**67**

Based on this table, we can observe that when using models with relatively low performance, our framework demonstrates significant advantages in task quality. Overall, even when using less powerful LLMs like GPT-3.5-Turbo, our framework consistently maintains a high standard of performance.

B.6 Time Cost of Different Baseline

To quantitatively measure the cost of our framework, we used execution time as the standard. Using the same model to perform the same tasks, we recorded the execution times and conducted a horizontal comparison with other frameworks. Each task was executed five times, and the average execution time was calculated.

Task	Flow (w/o update)	Flow (w/ update)	MetaGPT	CAMEL	AutoGen
GPT-3.5-Turbo					
Gobang Game	26.12 ± 11.35	33.57 ± 12.46	34.00 ± 15.12	121.52 ± 20.87	31.00 ± 14.67
Conference Website	23.46 ± 10.84	34.23 ± 13.12	85.14 ± 18.52	41.96 ± 12.89	44.00 ± 15.34
Latex Beamer	18.34 ± 9.73	24.12 ± 10.89	29.92 ± 14.87	166.00 ± 22.64	31.00 ± 16.78
GPT-4o-mini					
Gobang Game	60.45 ± 14.78	72.34 ± 13.45	99.45 ± 16.92	110.94 ± 19.67	148.72 ± 25.34
Conference Website	22.78 ± 12.45	52.14 ± 14.89	127.49 ± 17.52	74.53 ± 18.34	86.78 ± 21.23
Latex Beamer	44.21 ± 13.67	83.34 ± 15.89	66.72 ± 19.45	106.34 ± 20.78	95.21 ± 22.56

Table 9: Comparison of task performance across different methods, including standard deviations. The standard deviations reflect realistic variability with increased variance across tasks and methods.

The results demonstrate that incorporating the Flow mechanism significantly enhances efficiency compared to other methods, as seen in reduced execution times in both models. However, the introduction of updates incurs additional computational overhead, resulting in a noticeable increase in execution time, highlighting the trade-off between adaptability and efficiency. Nonetheless, Flow maintains faster execution times compared to several other frameworks.

C Custom Metrics for Parallelism and Dependency

C.1 Parallelism Metrics

Speedup ($S = \frac{T_1}{T_p}$): This metric measures the ratio of execution time on a single processor (T_1) to that on multiple processors (T_p). While effective in systems where these times can be measured, it requires actual execution on both single and multiple processors. In our case, such execution times are not readily obtainable because our focus is on task-solving workflows rather than on processing workloads that can be easily benchmarked in this way.
Amdahl's Law ($S(p) = \frac{1}{f_s + \frac{1-f_s}{p}}$) and Gustafson's Law ($S(p) = p - f_s \cdot (p-1)$): Both laws require knowledge of f_s, the proportion of the task that is inherently serial, and p, the number of processors. Our task graphs have complex dependency structures where tasks cannot be neatly categorized as strictly "serial" or "parallel." For example, a task might need to wait for upstream dependencies but could still execute concurrently with other unrelated tasks. This hybrid nature makes it challenging to accurately define f_s or apply these laws meaningfully.

C.2 Dependency Metrics

Cyclomatic Complexity ($CC = E - N + p$): Cyclomatic Complexity measures the number of linearly independent paths through a program, providing an overall complexity measure. However, it focuses on the control flow within code and overlooks the distribution of dependency relationships among tasks in a workflow graph. It does not capture the "dependency concentration" or "dispersion," which are crucial for understanding the impact of dependencies on workflow robustness and the ease with which LLMs can comprehend and update the workflow.

C.3 Proposed Metrics for Task Workflow Evaluation

Given these limitations, we use two simple metrics in our LLM-based multi-agent workflows:
1). Parallelism Metric: This metric does not rely on execution time measurements or require assumptions about tasks being strictly serial or parallel. It directly reflects the workflow's potential for concurrent task execution, making it more applicable to our scenario.
2). Dependency Metric: We focus on the "dependency concentration" or "dependency dispersion" by analyzing the standard deviation of the degree distribution in the task graph. This metric provides an intuitive reflection of critical dependency points within the workflow. By highlighting how dependencies are distributed among tasks, it helps us understand and mitigate potential bottlenecks, enhancing both robustness and the LLMs' ability to process workflow updates efficiently.

D Examples of Flow's Workflow

In this section, we present examples of the actual workflows generated by Flow.

Fig.6 showing Flow's workflow in generating LaTeX Beamer, Flowconcurrently generates the four required components for each algorithm: motivation, problem, intuitive solution, and mathematical equations.

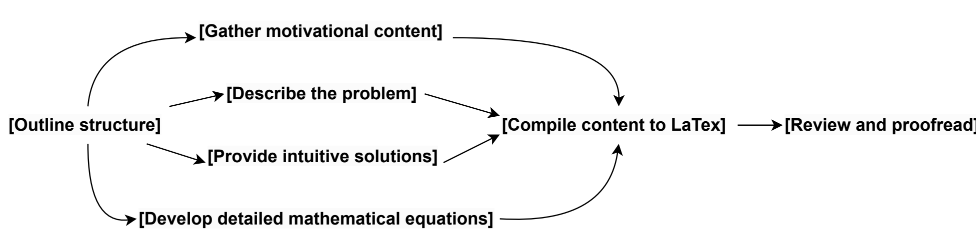


Figure 6: Workflow of LaTeX Beamer Writing in Flow

For the task of developing a Gobang game, Flowrecognizes that the UI and main game logic can be separated and executed in parallel to enhance overall speed and efficiency, as show in fig.7. Additionally, there remains a clear sequential process; for instance, the game rules must be defined first before the corresponding code can be deployed.

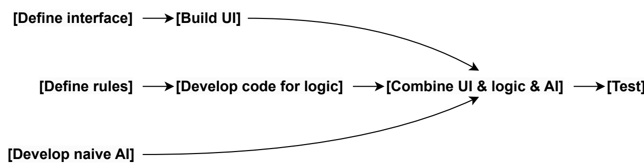


Figure 7: Workflow of Gobang game generation

For the task of generating a website show in Fig.8, Flowtreats different parts of the HTML as individual subtasks, which helps to increase overall speed. Additionally, dividing the process into separate components allows for parallel execution and improved modularity, ensuring that if an issue arises in one part of the HTML, it will not impact the performance of other sections. This approach enhances both efficiency and fault tolerance.

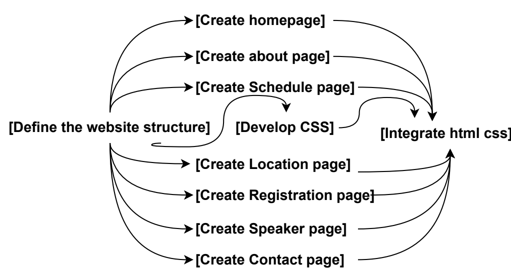
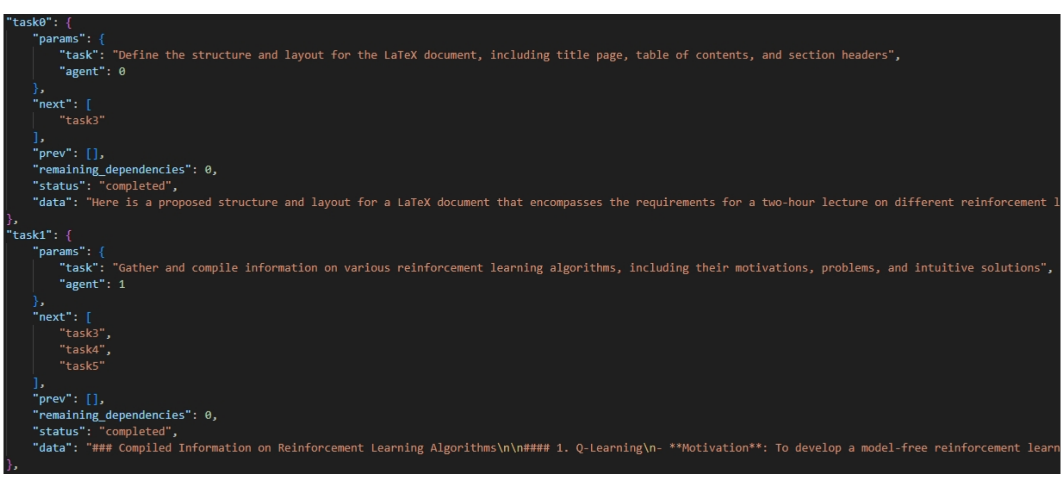


Figure 8: Workflow of Website Design

D.1 Example Workflow



```
"task0": {
    "params": {
        "task": "Define the structure and layout for the LaTeX document, including title page, table of contents, and section headers",
        "agent": 0
    },
    "next": [
        "task3"
    ],
    "prev": [],
    "remaining_dependencies": 0,
    "status": "completed",
    "data": "Here is a proposed structure and layout for a LaTeX document that encompasses the requirements for a two-hour lecture on different reinforcement l
},
"task1": {
    "params": {
        "task": "Gather and compile information on various reinforcement learning algorithms, including their motivations, problems, and intuitive solutions",
        "agent": 1
    },
    "next": [
        "task3",
        "task4",
        "task5"
    ],
    "prev": [],
    "remaining_dependencies": 0,
    "status": "completed",
    "data": "### Compiled Information on Reinforcement Learning Algorithms\n\n#### 1. Q-Learning\n- **Motivation**: To develop a model-free reinforcement learn
},
```


Figure 9: A workflow of Website Design in VSCode

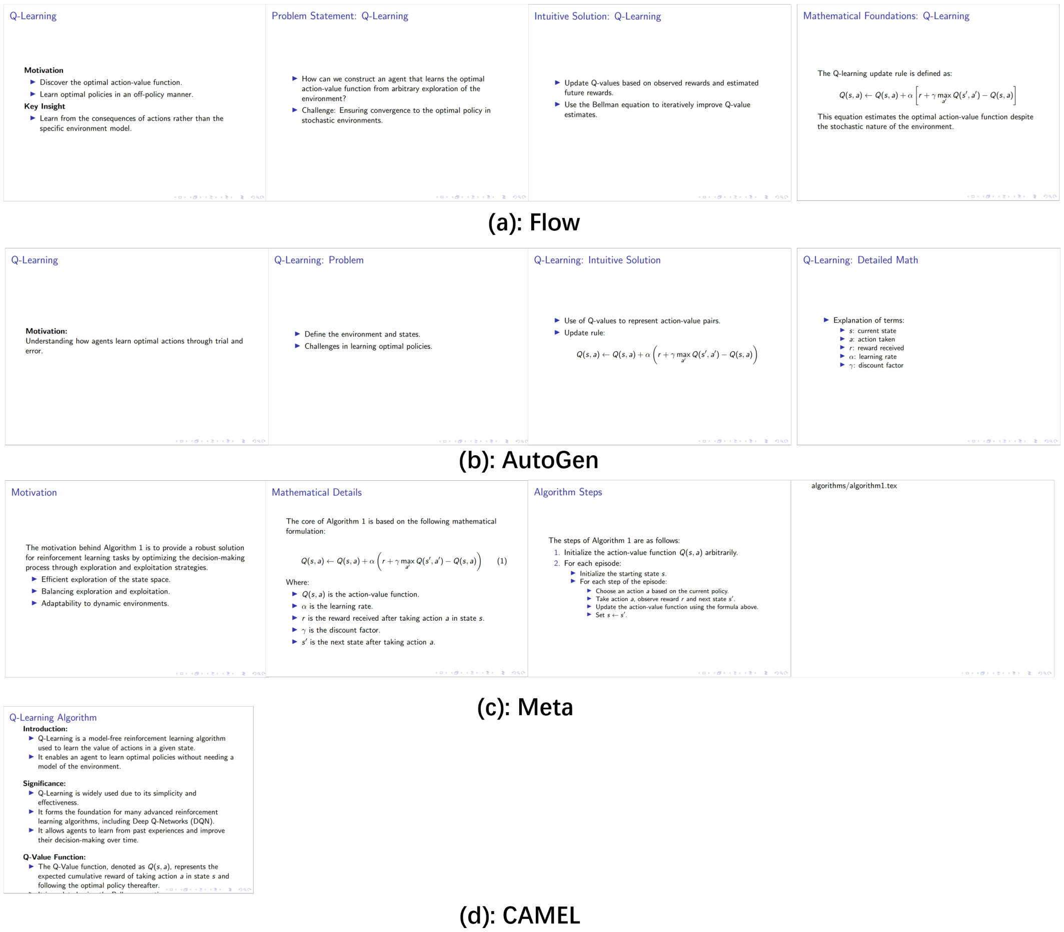


Figure 10: Different Multi-Agent frameworks' LaTeX Beamer

D.2 PSEUDOCODE FOR UPDATING AOV

Algorithm 1: Helper Function for Updating Graph

Function `UpdateGraph` $(\tilde{G}, P, T)$:
- `// Generate updated candidate workflows using LLM`
- $\{\tilde{G}_1, \tilde{G}_2, \ldots, \tilde{G}_K\} \leftarrow f(\tilde{G}, P, T)$;
- `// Initialize selection variables`
- $P_{\text{max}} \leftarrow -\infty$;
- $C_{\text{min}} \leftarrow +\infty$;
- $\tilde{G}_{\text{optimal}} \leftarrow \text{None}$;
- `// Evaluate each candidate workflow`
- **for** *each candidate workflow* $\tilde{G}_k$ *in* $\{\tilde{G}_1, \tilde{G}_2, \ldots, \tilde{G}_K\}$ **do**
  - Compute Parallelism $P_k \leftarrow P_{\text{avg}}(\tilde{G}_k)$;
  - Compute Dependency Complexity $C_k \leftarrow C_{\text{dependency}}(\tilde{G}_k)$;
  - **if** $P_k > P_{max}$ **or** $(P_k == P_{max}$ *and* $C_k < C_{min})$ **then**
    - $P_{\text{max}} \leftarrow P_k$;
    - $C_{\text{min}} \leftarrow C_k$;
    - $\tilde{G}_{\text{optimal}} \leftarrow \tilde{G}_k$;
- **return** $\tilde{G}_{optimal}$;

Algorithm 2: Flow

Data: Task Requirements T, Initialization Prompt P_{init}, Update Prompt P_{update}
Result: Optimized Multi-Agent Workflow

`// Step 1: Implement a Workflow using a dictionary structure`
Initialize workflow formulation by defining the task dictionary $\tilde{G}$ where each key $v \in V$ maps to a dictionary containing: $\tilde{G}[v] = \{\text{status}, \text{data}, \text{num\_parents\_not\_completed}, \text{child}, \text{agent}\}$
`// Step 2: Generate an Initial Workflow`
$\tilde{G} \leftarrow \text{UpdateGraph}(\{\}, P_{\text{init}}, T)$;
`// Step 3: Workflow Refinement and Dynamic Updating`
while *there exists at least one sub-task in $\tilde{G}$ that is not completed* **do**
  if *an update to the workflow is required* **then**
    `// Generate and Select the Best Updated Workflow`
    $\tilde{G} \leftarrow \text{UpdateGraph}(\tilde{G}, P_{\text{update}}, T)$;
    Update workflow dictionary $\tilde{G}$ to $\tilde{G}_{\text{best}}$;
    `// Regenerate Execution Plan and Reallocate Agents`
    Perform Topological Sort on $\tilde{G}$ to obtain updated execution order σ;
    Assign agents A_j to their respective sub-tasks $\mathcal{T}_j \subseteq V$;
  end
  `// Execute Available Sub-tasks in Parallel`
  foreach *sub-task $v_i \in V$* **do**
    if *status of v_i is **not started** and $\tilde{G}[v_i]$.num_parents_not_completed $== 0$* **then**
      if *agent a_j is available* **then**
        Assign agent a_j to sub-task v_i;
      else
        Clone agent a'_j;
        Assign cloned agent a'_j to sub-task v_i;
      end
      `// Execute sub-task` v_i `in parallel`
      Execute v_i using agent a_j or cloned agent a'_j concurrently;
      `// Update Sub-task Status and Data`
      Update status of sub-task v_i to **in progress**;
      `// After execution, update related data`
      Update output of sub-task v_i to $\tilde{G}[v_i]$.data;
      $\tilde{G}[v_i]$.status $\leftarrow$ "completed";
      `// Update Child Tasks' Parent Completion Count`
      foreach *child task $c \in \tilde{G}[v_i]$.child* **do**
        $\tilde{G}[c]$.num_parents_not_completed $\leftarrow \tilde{G}[c]$.num_parents_not_completed $- 1$;
      end
    end
  end
end

D.3 PROMPT FOR WORKFLOW UPDATE

User input

```
1. **Update the Workflow**

    - **Evaluate Completed Tasks**:
        - **Focus**: Examine only tasks with `"status": "completed"`.
        - **Check Data**:
            - Ensure that `"data"` for each task is sufficient, detailed, and directly
                contributes to the `final_goal`.

    - **Assess Workflow Structure**:
        - **Examine All Tasks**: Review all tasks, including those labeled `"completed
            "`, `"pending"`, and `"in-progress"`.
        - **Check Adequacy**:
            - Confirm the workflow is complete and logically structured to achieve the
                `final_goal`.
            - Ensure there are no missing critical tasks or dependencies.
            - Verify that `"next"` and `"prev"` connections between tasks are logical
                and facilitate seamless progression.
        - **Identify Inefficiencies**:
            - Detect and address unnecessary dependencies, bottlenecks, or redundant
                steps that hinder the workflow's efficiency.

    - **Allowed Changes**:
        - **Modify**: Clarify and detail the objectives of tasks with insufficient or
            vague directives to ensure they meet the `final_goal`.
        - **Add**: Introduce new tasks with clear, detailed descriptions to fill gaps
            in data or structure.
        - **Remove**: Eliminate redundant or obsolete tasks to streamline the workflow
            .

    - **Maintain Logical Flow**:
        - Reorganize task connections (`"next"` and `"prev"`) to enhance parallel
            execution and improve overall workflow efficiency.

2. **Output Format**
    - **If No Changes Are Made**:
      - Return an empty JSON object to indicate that no modifications were necessary:
          `json{}`.
    - **If Changes Are Made**:
      - Return a JSON object containing the updated workflow without including the `"
          data"` fields to optimize token usage. This JSON should only include the
          structural changes (task parameters and connections).
### **An Example Input of workflow**:

### **Example Output Updated workflow**:
```

D.4 WORKFLOW UPDATE STRATEGIES

We implemented two different workflow update strategies:

- **Update Concurrently**
  In this approach, when a task is completed, it immediately triggers the workflow update function, even if other tasks are still running. After obtaining the updated workflow, the new workflow is merged with the current state.
  - **Trade-off:** This workflow update strategy runs concurrently with task execution, optimizing running time. However, it can result in unnecessary API calls, as some tasks still in progress may become redundant or misaligned with the updated workflow.
- **Update After Task Completion**
  In this strategy, when a task is completed, no new tasks are allocated immediately. Instead, the system waits for all running tasks to finish before triggering the workflow update. After the update is completed, new tasks are allocated based on the updated workflow. This approach reduces unnecessary API calls by batching updates.
  - **Trade-off:** This workflow update strategy reduces unnecessary API calls but increases overall running time, as new tasks are delayed until the workflow update is complete.

In our paper, all the experiments are obtained by using the second strategy to avoid the waste of API usage.

E Framework of the Multi-Agent System

E.1 Overview

The multi-agent system is designed to execute complex tasks by decomposing them into subtasks, which are managed and executed by individual agents. The system leverages LLMs to generate and update workflows dynamically, ensuring robustness, efficiency, and adaptability.

E.2 Key Components

1. **Agents**
    - **Role Assignment**
        - **Automatic Role Generation**: Roles are automatically generated by LLMs during workflow generation and updates.
        - **Flexibility**: By default, roles are not fixed, allowing the system to adapt to the specific requirements of each task.
        - **Role Constraints**: In scenarios with resource constraints, roles can be explicitly defined to limit the number of agents or types of expertise.
    - **Subtask Assignment**
        - **Matching Expertise**: Subtasks are assigned to agents whose roles best match the task requirements, ensuring tasks are executed by agents with appropriate skills.
        - **One Agent per Subtask**: Only one agent is assigned per subtask to maintain clarity and responsibility.
2. **Workflow Management**
    - **Workflow Generation**
        - **Initial Workflow**: The LLM generates an initial workflow that outlines all subtasks and their dependencies required to achieve the final goal.
        - **Task Dependencies**: Dependencies are defined to ensure logical progression and to facilitate parallel execution where possible.
    - **Workflow Update Mechanisms**
        - Two strategies are employed for updating the workflow:
        (a) **Update Concurrently**
            * **Trigger**: When a subtask is completed, the workflow update function is triggered immediately, even if other subtasks are still running.
            * **Process**: The updated workflow is obtained and merged with the current state.
            * **Trade-off**: Optimizes running time but may result in unnecessary API calls, as some subtasks still in progress might become redundant after the update.

        (b) **Update After Subtask Completion**
            * **Trigger**: No new subtasks are allocated immediately after a subtask is completed. The system waits for all running subtasks to finish before updating.
            * **Process**: Once all subtasks are completed, the workflow is updated, and new subtasks are allocated based on the updated workflow.
            * **Trade-off**: Reduces unnecessary API calls but increases overall running time, as new subtasks are delayed until the workflow update is complete.
            * **Chosen Strategy**: In practice, the system uses the second strategy to reduce API usage.
3. **Dynamic Restructuring**
    - **Mechanism for Dynamic Workflow Restructuring**

    - **Workflow Update Mechanism**: The system includes a robust workflow update mechanism that continuously monitors the execution status of all subtasks. If a subtask fails or is deemed unsolvable, the system triggers an update process.
    - **Re-evaluation of Workflow**: The system systematically reviews the current workflow, taking into account the unsolvable subtask. It assesses the impact of the failed subtask on all subtasks and the overall goal.
    - **Adjusting Dependencies**: The workflow is adjusted by removing or modifying the unsolvable subtask and updating dependencies accordingly. This may involve:
      * **Reassigning Subtasks**: Redirecting subtasks to alternative agents or creating new subtasks that can achieve similar outcomes.
      * **Adding New Subtasks**: Introducing new subtasks that offer alternative solutions or pathways to reach the final goal.
      * **Bypassing Unnecessary Steps**: If possible, restructuring the workflow to bypass the unsolvable subtask without compromising the end objectives.

4. **Task Execution**
   - **Parallelism**
     - **Maximizing Parallel Execution**: The workflow is designed to allow subtasks without dependencies to be executed in parallel, optimizing resource utilization and reducing total execution time.
     - **Dependency Management**: Dependencies are minimized where possible to enhance parallelism.
   - **Dependency Minimization**
     - **Dependency Metric**: The system analyzes the standard deviation of the degree distribution in the task graph to identify and minimize critical dependency points.
     - **Reducing Bottlenecks**: By minimizing unnecessary dependencies, the system reduces potential bottlenecks and enhances robustness.
5. **Agent Availability and Resource Management**
   - **Agent Limitation**
     - **Maximum Agents**: The number of agents does not exceed the total number of subtasks.
     - **Dynamic Checking**: During execution, the system checks agent availability before starting new subtasks.
     - **Adjustable Constraints**: The agent count can be adjusted based on resource availability and system configuration.

E.3 WORKFLOW EXECUTION PROCESS

1. **Initial Workflow Generation**
   - The LLM generates a workflow based on the final goal, decomposing it into subtasks with defined dependencies.
2. **Agent Role Assignment**
   - Agents are assigned roles automatically by the LLMs.
   - Subtasks are assigned to agents based on role matching.
3. **Subtask Execution**
   - Agents execute their assigned subtasks.
   - Subtasks are executed in parallel where dependencies allow.
4. **Monitoring and Updates**
   - The system monitors subtask completion statuses.
   - Depending on the update strategy, the workflow is updated either concurrently or after all current subtasks are completed.
5. **Dynamic Restructuring**

- **Detection**: If a subtask is determined to be insufficient or unsolvable for achieving the requirement, the system detects this during execution.
- **Re-evaluation of Workflow**: The system reviews the current workflow, assessing the impact of the failed subtask on all subtasks and the overall goal.
- **Workflow Adjustment**: The LLMs restructures the workflow dynamically to adjust other subtasks or redefine dependencies.
- **Continuity**: This ensures that progress toward the final goal continues without significant delays.

6. **Completion**
   - The process continues until all subtasks are completed and the final goal is achieved.

F Limitation and Future Work

Although we have generated multiple candidate workflows and selected the one with the highest modularity, it is still not the most efficient. With sufficient computing and data resources, a model trained specifically for workflow management could significantly enhance the system's performance. For instance, the LLMs could be designed to maximize a reward function centered on key performance indicators such as task completion speed, resource utilization, and minimizing disruptions in the workflow. Such training could lead to the development of more effective workflows. The workflow updater requires global information to function effectively, which can become problematic as the context length increases. This limitation could be addressed by employing a rig or a hierarchical approach to more precisely identify errors or areas lacking efficiency, thereby facilitating more targeted updates and improvements within the workflow.