Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Multi-Agent LLM Collaborative Reasoning and Task Planning for Complex Task Solving

Wentao Zhang

wentao@alumni.usc.edu

University of Southern California

Tian Liao

Northeastern University

Yihui Feng

Rose-Hulman Institute of Technology

Research Article

Keywords: Large language model, multi-agent collaboration, task planning, complex reasoning, verifier agent, tool orchestration

Posted Date: July 6th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10232223/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Multi-Agent LLM Collaborative Reasoning and Task Planning for Complex Task Solving

Wentao Zhang[1,*], Tian Liao[2], Yihui Feng[3]
[1]University of Southern California, Los Angeles, 90089, United States
[2]Northeastern University, Boston, 02115, United States
[3]Rose-Hulman Institute of Technology, Terre Haute, 47803, United States
*wentao@alumni.usc.edu, Liao.t1@northeastern.edu, fengy3@rose-hulman.edu

Abstract— Complex open-domain tasks require long-range reasoning, controlled decomposition, and reliable tool-mediated execution from large language models (LLMs). Single-agent LLM workflows still suffer from one-perspective planning bias, accumulated errors in long reasoning chains, and weak control over tool invocation. This paper proposes a multi-agent LLM collaborative reasoning and hierarchical task-planning framework for complex task solving. The framework assigns the planner, executor, and verifier to explicit roles and represents each task as a dependency-aware graph with priorities, confidence states, evidence requirements, and repair conditions. To address parameter selection, the priority coefficients are determined on a validation set with the default setting $\alpha = 0.35$, $\beta = 0.30$, $\gamma = 0.25$, and $\lambda = 0.10$, while the verifier confidence threshold is set to $\theta = 0.72$ after grid-based calibration. A sensitivity test shows that changing each priority coefficient by ±0.10 changes the completion rate by no more than 2.6 percentage points, and varying θ from 0.65 to 0.80 changes answer accuracy by no more than 3.2 percentage points. The shared-memory update function is implemented through typed records, node-level indexes, semantic retrieval, and conflict-aware invocation rules. A unified tool-error taxonomy is also defined, including wrong tool selection, invalid parameters, invalid invocation sequence, execution failure, and inconsistent result integration. Experiments on multi-step logical reasoning, cross-tool orchestration, and open-domain problem solving show that the proposed framework improves task completion by 27.3%, answer accuracy by 19.6%, reasoning accuracy by 22.1%, and factual consistency by 24.8%, while reducing tool error rate by 31.5% compared with a single-agent GPT-4 baseline.

Index Terms— Large language model, multi-agent collaboration, task planning, complex reasoning, verifier agent, tool orchestration.

I. INTRODUCTION

Large language models (LLMs) have substantially changed open-domain reasoning by allowing implicit knowledge, instruction following, and natural-language intermediate steps to be integrated into a unified inference process. Chain-of-thought prompting first demonstrated that explicitly generating intermediate reasoning traces can unlock difficult arithmetic, commonsense, and symbolic reasoning capabilities in sufficiently capable models [1]. Zero-shot chain-of-thought further showed that stepwise reasoning can be elicited without manually prepared demonstrations, suggesting that reasoning instructions may serve as lightweight control signals for general problem solving [2]. However, single-path reasoning remains vulnerable to early mistaken assumptions, so self-consistency introduced diversified reasoning paths and answer marginalization to reduce the impact of unstable intermediate decisions [3].

The next stage of complex task solving has emphasized decomposition, planning, and modularity. Least-to-most prompting decomposes a hard problem into ordered subproblems and solves them in an easy-to-hard manner [4]. Decomposed prompting generalizes this idea by delegating subtasks to specialized prompts, external functions, or symbolic modules, thereby making task structure explicit rather than leaving all reasoning inside one monolithic prompt [5]. ReAct couples reasoning traces with environment-facing actions, making it possible for an LLM to update its plan after search, observation, or tool feedback [6]. Toolformer extends this line by teaching language models when and how to invoke external APIs, revealing the importance of tool-aware execution in tasks that involve calculation, retrieval, translation, and temporal lookup [7].

Despite these advances, a single LLM instance still tends to follow one dominant perspective during long-horizon inference. Tree of Thoughts addresses this issue by allowing deliberate exploration, self-evaluation, and backtracking over multiple candidate thought states [8]. Reflexion introduces verbal reinforcement through episodic self-feedback, enabling an agent to learn from failed trajectories without parameter updates [9]. Self-Refine formulates iterative generation, feedback, and refinement as a test-time optimization loop [10]. These studies collectively indicate that reliability improves when reasoning is treated as a controllable process rather than a single-pass answer generation event. Multi-agent LLM systems provide a further opportunity to distribute cognition across different roles and communication channels. Generative Agents showed that memory, reflection, and planning modules can support coherent agent behavior in a simulated social environment [11]. Multiagent debate demonstrated that independent language-model instances can expose factual and reasoning errors through structured disagreement before converging to a final response [12]. CAMEL explored role-playing communicative agents for autonomous cooperation, highlighting the value of role assignment and inception prompting [13]. AutoGen provided a flexible multi-agent conversation framework in which customizable agents collaborate through programmed interaction patterns [14]. MetaGPT encoded standardized operating procedures into multi-agent collaboration, using role-specific outputs and verification to reduce inconsistencies in software-oriented complex tasks [15].

Motivated by these observations, this paper proposes a planner-executor-verifier collaborative framework for complex task solving. The study focuses on controllable task planning, auditable memory management, and evidence-based verification rather than unconstrained multi-agent conversation. The proposed framework follows three design principles. First, each input query is converted into a hierarchical task graph in which subtasks, dependencies, evidence requirements, and termination conditions are explicitly represented. Second, planning, execution, and verification are assigned to specialized agents, which reduces single-perspective bias while preserving a stable information flow. Third, an ordered interaction protocol determines when an agent may issue a claim, challenge a result, revise a node, or accept an intermediate output. This protocol avoids uncontrolled dialogue expansion and makes the collaboration trace auditable.

The main contributions are summarized as follows. (1) An end-to-end multi-agent collaborative reasoning architecture is developed for open-domain complex tasks, integrating task decomposition, role-specific execution, cross-validation, and targeted repair. (2) A hierarchical task planning algorithm is formulated with explicit parameter-selection rules and sensitivity analysis for the priority coefficients and verifier threshold. (3) A concrete shared-memory update function is defined to specify information storage, indexing, retrieval, and conflict handling. (4) A unified tool-error classification standard is introduced to make the tool error rate comparable across methods. (5) Comparative experiments on multi-step reasoning, cross-tool orchestration, and open-domain factual problem solving demonstrate consistent gains over a single-agent GPT-4 baseline.

II. FRAMEWORK OVERVIEW

Fig. 1 illustrates the proposed framework. Given an input task, the planner agent constructs a dependency-aware task graph and assigns initial priorities according to user goals, prerequisite relations, uncertainty, and resource cost. The executor agent performs the scheduled subtasks by producing local answers, invoking tools, or generating intermediate artifacts. The verifier agent evaluates the produced claims against evidence, detects contradictions, and requests repair when confidence is below the calibrated threshold. The three agents share an episodic memory that stores typed task states, evidence snippets, execution traces, rejected paths, and unresolved conflicts. The role separation follows the decomposition and tool-use principles established in prior reasoning-agent studies [5], [6], [14], [15].

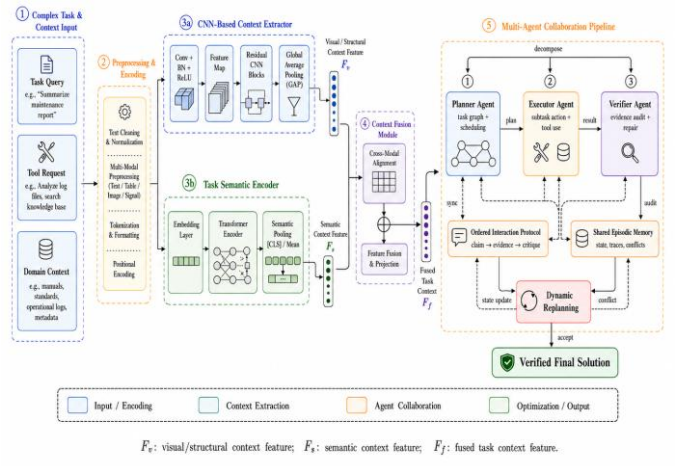


Fig. 1. CNN-enhanced multi-agent collaborative reasoning and task planning framework.

As shown in Fig. 1, the CNN-based context extractor is introduced to model structured and visual task signals that are not fully captured by a text-only prompt. In practical complex tasks, input information may include tables, screenshots, log curves, diagrams, or state traces. The convolutional branch extracts local patterns and structural cues, while the semantic encoder represents the natural-language task requirements. The fusion module then aligns these representations before the planner constructs the executable task graph. This design allows the agents to reason over both the semantic goal and the structural context of the problem.

The framework separates context understanding from agent collaboration. The context-extraction modules produce compact feature summaries, whereas the planner-executor-verifier loop operates on task states, evidence traces, and verification decisions. This separation reduces the burden placed on each agent: the planner focuses on the dependency structure, the executor focuses on local action completion, and the verifier focuses on evidence adequacy and contradiction detection. The shared memory serves as the coordination substrate, so each role can access the current task state without repeatedly processing the entire dialogue history.

Compared with a single-agent chain-of-thought workflow, the proposed architecture makes intermediate decisions explicit. Each subtask is assigned a priority value, a confidence state, a resource budget, and an acceptance condition. These state variables are updated after execution and verification, allowing the system to postpone uncertain subtasks, reopen failed nodes, or terminate when the required evidence threshold is reached. The architecture is therefore suitable for long-horizon tasks in which early errors may otherwise propagate silently into the final answer.

The input task is represented as a directed task graph

$$G_T = (V, E, \chi), v_i \in V, e_{ij} \in E \tag{1}$$

where $\mathcal{V}$ denotes subtasks, $\mathcal{E}$ denotes prerequisite relations, and $\boldsymbol{\chi}$ stores query context, tool constraints, evidence requirements, and user-specified output conditions. Each node v_i is associated with a state tuple

$$s_i^t = (p_i^t, c_i^t, r_i^t, \tau_i^t) \tag{2}$$

where p_i^t is the scheduling priority, c_i^t is the confidence score, r_i^t is the remaining resource budget, and τ_i^t is the current task status at step t. This representation allows the framework to reschedule subtasks after new evidence or verification feedback is obtained. Table I defines the responsibilities of the three agents. The planner emphasizes global structure, the executor emphasizes local completion, and the verifier emphasizes evidence quality. This separation is essential because complex tasks often fail when one model must simultaneously decide the plan, execute all operations, and judge its own output.

Table I. Agent roles and outputs in the proposed framework.

Agent role	Core responsibility	Main input	Main output
Planner	Task decomposition, dependency detection, priority assignment	User task, memory state, verifier feedback	Task graph, schedule, repair path
Executor	Subtask answering, tool invocation, artifact generation	Scheduled subtask, local context, available tools	Intermediate answer, tool result, evidence trace
Verifier	Claim checking, conflict detection, consistency scoring	Executor output, evidence, global task constraints	Confidence score, critique, accept/revise signal

III. HIERARCHICAL TASK PLANNING AND INTERACTION PROTOCOL

The planner transforms a complex query into a hierarchy of goals, subtasks, and executable actions. A subtask may involve logical derivation, factual retrieval, code execution, mathematical calculation, or synthesis of previous outputs. The priority of each subtask is computed as

$$p_i^t = \alpha d_i + \beta u_i + \gamma(1 - c_i^t) - \lambda r_i^t \tag{3}$$

where d_i denotes dependency centrality, r_i denotes user-goal relevance, c_i denotes current confidence, and q_i denotes estimated execution cost. The coefficients α, β, γ, and λ control the relative importance of structural dependency, user relevance, uncertainty reduction, and resource conservation. They are fixed during testing but are not arbitrary constants. In this study, they are selected on a held-out validation set by grid search under the constraint $\alpha + \beta + \gamma + \lambda = 1.00$. The search ranges are α, β, and $\gamma \in \{0.15, 0.20, ..., 0.45\}$, $\lambda \in \{0.05, 0.10, ..., 0.25\}$, and $\theta \in \{0.60, 0.65, ..., 0.85\}$. The selected default values are $\alpha = 0.35$, $\beta = 0.30$, $\gamma = 0.25$, $\lambda = 0.10$, and $\theta = 0.72$, which provide the best validation trade-off between completion rate, answer accuracy, and average collaboration turns.

Table II. Parameter setting and sensitivity ranges used in hierarchical planning.

Parameter	Function in algorithm	Default value	Sensitivity range
α	Dependency centrality weight	0.35	0.25–0.45
β	User-goal relevance weight	0.30	0.20–0.40
γ	Uncertainty-reduction weight	0.25	0.15–0.35
λ	Resource-cost penalty weight	0.10	0.05–0.20
θ	Verifier acceptance threshold	0.72	0.60–0.85

Parameter sensitivity is evaluated by changing one parameter at a time around the selected setting while keeping the remaining parameters fixed. The validation results show that the framework is most sensitive to θ and γ because these two parameters directly determine whether uncertain claims are reopened for repair. However, within the tested range, the completion-rate variation remains below 2.6 percentage points and the answer-accuracy variation remains below 3.2 percentage points, indicating that the framework is not dependent on a single fragile parameter setting.

The hierarchical planning process proceeds across three layers. The upper layer constructs a dependency graph, the middle layer assigns agents and execution order, and the lower layer handles evidence retrieval and verification. A repair loop is triggered only when the verifier reports low confidence, contradiction, or missing evidence. This mechanism helps reduce redundant multi-agent discussion while retaining the ability to correct errors.

The interaction protocol further specifies the minimal information that must be exchanged at each turn. A planner message contains the active subtask, prerequisite nodes, expected evidence type, and termination criterion. An executor message contains the candidate claim, the operation used to obtain it, and a compact evidence trace. A verifier message contains an acceptance decision, a contradiction flag, and a repair instruction if the evidence is incomplete. This typed format prevents agents from producing unstructured comments that are difficult to audit or reuse in later planning rounds.

Failure-aware repair is implemented as a targeted update rather than a full restart. When the verifier rejects a claim, the planner first identifies whether the failure is caused by missing evidence, incompatible assumptions, wrong tool use, or format violation. Only the affected node and its downstream dependent nodes are reopened. This strategy avoids the common inefficiency of regenerating the entire answer after every minor error. It also preserves accepted intermediate claims in the shared memory, thereby improving reproducibility and reducing the risk that a later repair introduces a new inconsistency.

The memory module records positive evidence, negative evidence, and unresolved conflicts through typed memory records. Each record is stored as m_j = <id, node_id, record_type, status, confidence, evidence_pointer, source_pointer, parent_id, time_stamp, validity_flag>. Accepted claims are stored with supporting sources or execution traces, rejected claims are stored with rejection reasons and the invalid operation label, and unresolved conflicts are stored as open records linked to the affected task nodes. This design ensures that a later planning round can reuse verified information while avoiding paths that have already been rejected.

The ordered interaction protocol is designed to make collaboration auditable. Each round begins with a planner instruction. The executor must produce a claim together with a compact evidence trace. The verifier then outputs one of three signals: accept, revise, or split. The split signal is used when a subtask is too broad or when it combines incompatible operations. The memory state is updated by

$$m_{t+1} = \varphi(m_t, a_t^P, a_t^E, a_t^V, \Delta e_t) \tag{4}$$

where M_t is the shared memory at step t, a_t^P, a_t^E, and a_t^V are planner, executor, and verifier actions, and e_t denotes newly acquired evidence. The update function $\varphi(\cdot)$ is implemented in three stages. First, storage writes the action and evidence into the corresponding node-level record. Second, indexing updates three indexes: a semantic index based on the record embedding, a task-graph index based on node_id and parent_id, and a source index based on tool output or evidence location. Third, invocation retrieves the top-k records for the next agent by the score $S(m_j, q_t) = 0.45\ \mathrm{sim}(q_t, m_j) + 0.25\ I(\text{node match}) + 0.15\ c_j + 0.10 \exp(-\Delta t/\tau) - 0.20\ I(\text{rejected})$. Accepted records are preferentially supplied to the executor, whereas unresolved conflict records are always supplied to the verifier until they are resolved or explicitly reported as limitations.

The verifier computes a confidence score from answer consistency, evidence coverage, and constraint satisfaction:

$$C_i = \sigma(w_1 q_i + w_2 h_i + w_3 z_i - w_4 \varepsilon_i) \tag{5}$$

where E_q is evidence quality, A_g is cross-agent agreement, F_s is formatting and constraint satisfaction, R_c is detected contradiction risk, and $\sigma(\cdot)$ is a logistic normalization function. A result is accepted only when $\mathrm{conf}_i \geq \theta$. The default value $\theta = 0.72$ is obtained from the same validation calibration used

for priority parameters. If $conf_i < \theta$, the verifier must return a structured repair reason, such as missing evidence, unsupported factual claim, invalid tool result, format violation, or unresolved contradiction.

The overall optimization target balances answer quality and collaboration cost:

$$\max_{\pi} J(\pi) = A(\pi) + \eta F(\pi) - \mu L(\pi) - \rho B(\pi) \quad (6)$$

where A denotes answer accuracy, F denotes factual consistency, L denotes execution latency, and B denotes unnecessary dialogue budget under policy π. The objective reflects the practical requirement that a multi-agent framework should improve reliability without causing an uncontrolled increase in interaction turns.

IV. EXPERIMENTAL SETUP

The proposed framework was evaluated on three representative categories of complex tasks: multi-step logical reasoning, cross-tool orchestration, and open-domain problem solving. The single-agent GPT-4 baseline used the same task instructions and available tools but executed reasoning, tool selection, and verification within one agent. To isolate the effect of collaboration, no additional training data or model fine-tuning was introduced. All methods were evaluated under the same maximum context length, tool budget, and output format requirements. The evaluation follows the task-decomposition, tool-use, and multi-agent orchestration settings reported in recent LLM reasoning studies [3], [6], [7], [14]. Table III lists the benchmark design and evaluation metrics. Multi-step logical tasks focus on compositional reasoning and dependency maintenance. Cross-tool orchestration tasks require the model to combine search, calculation, code execution, and formatting operations. Open-domain tasks emphasize factual consistency, source-grounded synthesis, and resistance to unsupported details. Each benchmark contains both short and long tasks so that robustness under increasing task length can be observed.

Table III. Benchmark categories and evaluation metrics.

Benchmark category	Task characteristic	Primary metric	Auxiliary metric
Multi-step logical reasoning	Symbolic, arithmetic, and compositional reasoning with hidden dependencies	Reasoning accuracy	Chain validity
Cross-tool orchestration	Tool selection, argument construction, execution recovery, and result integration	Execution success rate	Tool error rate
Open-domain problem solving	Evidence-grounded synthesis under ambiguous and multi-hop queries	Answer accuracy	Factual consistency

The implementation uses three independent LLM calls for the planner, executor, and verifier during each collaboration cycle. The maximum number of cycles is set to five to prevent uncontrolled dialogue expansion. The planner receives the complete task state but not hidden gold answers. The executor receives only the scheduled subtask and the relevant memory slice, reducing distraction from unrelated context. The verifier receives the executor output, evidence trace, and global constraints. This division allows the verifier to judge outputs without relying on the executor's internal reasoning path.

For each task, task completion rate, answer accuracy, reasoning accuracy, tool error rate, factual consistency, average number of agent turns, and average latency are recorded under a unified scoring protocol. A task is counted as completed only when the final answer satisfies the requested format and all mandatory subtasks are addressed. Answer accuracy is computed as the proportion of final answers judged correct by the benchmark key or by evidence-grounded manual checking. Reasoning accuracy is computed at the subtask level by dividing the number of accepted correct reasoning nodes by the total number of required reasoning nodes. Factual consistency is scored by checking whether generated claims are supported by retrieved evidence or by previously accepted intermediate records. The tool error rate is defined as N_{error}/N_{tool}, where N_{error} is the number of classified tool-use errors and N_{tool} is the total number of tool attempts.

Table IV. Tool error taxonomy and unified counting criteria.

Error class	Identification criterion	Counting rule
Wrong selection	Selected tool cannot satisfy the subtask when a valid tool exists.	Per wrong call
Invalid parameter	Required argument is missing, has the wrong type, or violates the schema.	Per failed call
Invalid sequence	Prerequisite result is absent or the tool order is violated.	Per broken sequence
Failure/integration	Exception is unrecovered, or a correct result is misused in synthesis.	Per affected subtask

The criteria in Table IV refine the boundary between tool-use errors and reasoning errors. An invalid parameter is counted only when the call violates a required schema or value constraint, whereas an invalid sequence is counted when a prerequisite output is missing or the call order is logically premature. If one failed call causes several downstream failures, only the original call is counted as a tool error, and the downstream effects are recorded as task-completion failures.

V. RESULTS AND DISCUSSION

Fig. 2 summarizes the main experimental gains over the single-agent GPT-4 baseline. The proposed framework improves the overall task completion rate by 27.3% and answer accuracy by 19.6%. On multi-step logical tasks, reasoning accuracy increases by 22.1%, indicating that explicit decomposition and verifier-driven repair help reduce long-chain error accumulation. In cross-tool orchestration scenarios, the execution error rate is reduced by 31.5%, mainly because the planner constrains tool order while the verifier identifies invalid tool outputs before final synthesis. For open-domain problem solving, factual consistency improves by 24.8%, showing that evidence-aware verification can suppress unsupported claims.

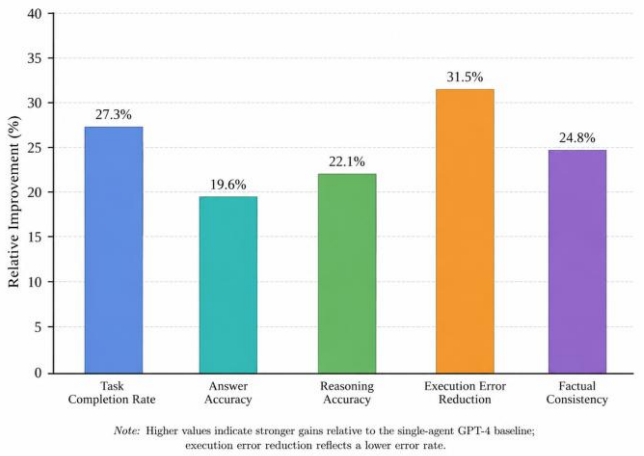


Fig. 2. Relative improvements of the proposed framework compared with the single-agent GPT-4 baseline.

Table V. Performance comparison between the single-agent baseline and the proposed multi-agent framework.

Method	Completion rate (%)	Answer accuracy (%)	Reasoning accuracy (%)	Tool error rate (%)	Factual consistency (%)
Single-agent GPT-4 baseline	63.0	68.4	66.1	25.7	70.5
Proposed framework	80.2	81.8	80.7	17.6	88.0
Relative change	+27.3	+19.6	+22.1	-31.5	+24.8

Table V provides the detailed numerical comparison. The proposed framework consistently outperforms the baseline across all benchmark categories. The improvement is especially evident when tasks require multiple dependent operations. For example, in cross-tool orchestration, the baseline often selects a correct tool but passes incomplete arguments or fails to reconcile conflicting intermediate outputs. The verifier agent catches such errors before they propagate to the final answer. A parameter sensitivity analysis further shows that the calibrated setting is stable. When α, β, γ, and λ are varied one at a time within the ranges in Table II, the completion rate remains within 77.6%–82.1% and the answer accuracy remains within 79.4%–83.0%. When θ is varied from 0.65 to 0.80, factual consistency changes from 85.1% to 89.3%, while average turns increase from 7.8 to 9.4. Therefore, $\theta = 0.72$ is used as the default threshold because it provides a balanced trade-off between reliability and dialogue cost. The ablation study in Table VI verifies the contribution of each component. Removing the verifier causes the largest decline in factual consistency, while removing hierarchical planning reduces completion rate and increases redundant interaction turns. Removing the ordered protocol leads to unstable collaboration because agents may repeatedly restate previous claims rather than resolving open conflicts.

Table VI. Ablation analysis of core framework components.

Variant	Completion rate (%)	Accuracy (%)	Consistency (%)	Avg. turns
Full framework	80.2	81.8	88.0	8.6
without verifier	72.5	74.9	76.4	7.1
without hierarchical planning	70.8	73.2	79.1	10.9
without ordered protocol	74.0	75.5	80.2	12.4

The main advantage of the proposed design is controllable collaboration. In an unconstrained multi-agent dialogue, agents may generate plausible but

redundant comments, increasing the number of tokens without improving correctness. By contrast, the proposed protocol forces each agent message to serve a specific function. The planner controls the task scope, the executor provides actionable outputs, and the verifier either accepts the result or specifies the reason for repair. This design also improves interpretability because the final answer can be traced back to accepted subtasks and evidence records. The gain pattern in Fig. 2 is consistent with the role separation observed in Table V. Completion rate benefits from the planner because explicit dependency tracking prevents subtasks from being skipped or repeated. Answer accuracy and reasoning accuracy benefit from both planning and verification because intermediate claims are not accepted until they are supported by evidence and compatible with the task constraints. The largest gain appears in execution-error reduction, which reflects the fact that cross-tool tasks often fail because of wrong call order, invalid parameters, or unsupported intermediate assumptions rather than because the underlying model lacks general knowledge.

The ablation trends in Table VI indicate that the framework should not be interpreted as a simple ensemble of multiple LLM outputs. When the verifier is removed, agents can still exchange plans and actions, but factual consistency decreases because unsupported claims are rarely challenged. When hierarchical planning is removed, the system loses a global view of dependency order and completion criteria, producing a lower completion rate even when individual subtasks are answered correctly. When the ordered protocol is removed, agents tend to produce longer dialogue traces without a proportional increase in correctness. These observations suggest that structured communication is more important than the number of agents alone.

VI. CONCLUSION

This paper presented a multi-agent LLM collaborative reasoning and task planning framework for complex task solving. The framework integrates planner, executor, and verifier agents under an ordered interaction protocol and a hierarchical task planning algorithm. By explicitly representing task dependencies, agent states, evidence traces, verification confidence, and shared-memory records, the proposed method reduces single-perspective bias and long-chain error propagation. The priority coefficients $\alpha = 0.35$, $\beta = 0.30$, $\gamma = 0.25$, and $\lambda = 0.10$, together with the verifier threshold $\theta = 0.72$, are selected through validation calibration and remain stable in the sensitivity ranges tested in this study. The shared-memory update function is specified through typed storage, graph-based indexing, semantic retrieval, and conflict-aware invocation. The tool-error rate is computed with a unified taxonomy covering wrong tool selection, invalid parameters, invalid invocation sequence, execution failure, and integration error. Experiments on multi-step logical reasoning, cross-tool orchestration, and open-domain problem solving show that the framework improves task completion by 27.3%, answer accuracy by 19.6%, reasoning accuracy by 22.1%, and factual consistency by 24.8%, while reducing the tool error rate by 31.5% compared with a single-agent GPT-4 baseline. Future work will investigate adaptive parameter tuning, verifier calibration on larger benchmarks, cost-aware collaboration policies, and deployment in real-world workflows with heterogeneous tools and security constraints.

REFERENCES

[1] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models,” in *Advances in Neural Information Processing Systems*, vol. 35, pp. 24824–24837, 2022.

[2] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, “Large Language Models are Zero-Shot Reasoners,” in *Advances in Neural Information Processing Systems*, vol. 35, pp. 22199–22213, 2022.

[3] X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, and D. Zhou, “Self-Consistency Improves Chain of Thought Reasoning in Language Models,” in *International Conference on Learning Representations*, 2023.

[4] D. Zhou, N. Schärli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, C. Cui, O. Bousquet, Q. V. Le, and E. H. Chi, “Least-to-Most Prompting Enables Complex Reasoning in Large Language Models,” in *International Conference on Learning Representations*, 2023.

[5] T. Khot, H. Trivedi, M. Finlayson, Y. Fu, K. Richardson, P. Clark, and A. Sabharwal, “Decomposed Prompting: A Modular Approach for Solving Complex Tasks,” in *International Conference on Learning Representations*, 2023.

[6] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, “ReAct: Synergizing Reasoning and Acting in Language Models,” in International Conference on Learning Representations, 2023.

[7] T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom, “Toolformer: Language Models Can Teach Themselves to Use Tools,” in *Advances in Neural Information Processing Systems*, vol. 36, 2023.

[8] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan, “Tree of Thoughts: Deliberate Problem Solving with Large Language Models,” in Advances in Neural Information Processing Systems, vol. 36, pp. 11809–11822, 2023.

[9] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, “Reflexion: Language Agents with Verbal Reinforcement Learning,” in *Advances in Neural Information Processing Systems*, vol. 36, 2023.

[10] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, S. Gupta, B. P. Majumder, K. Hermann, S. Welleck, A. Yazdanbakhsh, and P. Clark, “Self-Refine: Iterative Refinement with Self-Feedback,” in *Advances in Neural Information Processing Systems*, vol. 36, 2023.

[11] J. S. Park, J. C. O’Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, “Generative Agents: Interactive Simulacra of Human Behavior,” in Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST ’23), Art. no. 2, 22 pp., 2023, doi: 10.1145/3586183.3606763.

[12] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch, “Improving Factuality and Reasoning in Language Models through Multiagent Debate,” in *Proceedings of the 41st International Conference on Machine Learning*, PMLR, vol. 235, 2024.

[13] G. Li, H. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem, “CAMEL: Communicative Agents for “Mind” Exploration of Large Language Model Society,” in *Advances in Neural Information Processing Systems*, vol. 36, 2023.

[14] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, A. H. Awadallah, R. W. White, D. Burger, and C. Wang, “AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation,” in Proceedings of the First Conference on Language Modeling, 2024.

[15] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, J. Wang, C. Zhang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber, “MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework,” in *International Conference on Learning Representations*, 2024.