SCALING LARGE LANGUAGE MODEL-BASED MULTI-AGENT COLLABORATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent breakthroughs in large language model-driven *autonomous agents* have revealed that *multi-agent collaboration* often surpasses each individual through collective reasoning. Inspired by the neural scaling law—increasing neurons enhances performance, this study explores whether the continuous addition of collaborative agents can yield similar benefits. Technically, we utilize directed acyclic graphs to organize agents into a multi-agent collaboration network (MACNET), upon which their interactive reasoning is topologically orchestrated for autonomous task solving. Extensive evaluations reveal that it effectively supports collaboration among over a thousand agents, with irregular topologies outperforming regular ones. We also identify a *collaborative scaling law*—the overall performance follows a logistic growth pattern as agents scale, with collaborative emergence occurring earlier than traditional neural emergence. We speculate this may be because scaling agents catalyzes their multidimensional considerations during interactive reflection and refinement, thereby producing more comprehensive solutions.

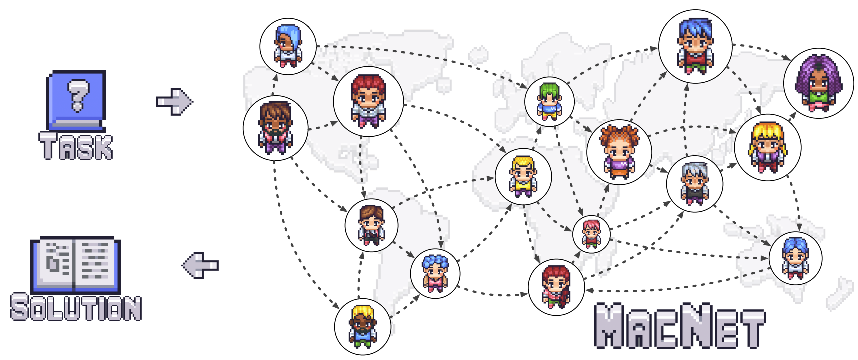


Figure 1: Multi-agent collaboration network (MACNET) uses directed acyclic graphs to arrange agents for collaborative interactions, facilitating autonomous task-solving through collective reasoning.

1 INTRODUCTION

In the rapidly advancing field of artificial intelligence, *large language models* (LLMs) have driven transformative shifts across numerous domains due to their remarkable linguistic capacity to seamlessly integrate extensive world knowledge (Vaswani et al., 2017; Brown et al., 2020). Central to this breakthrough is the *neural scaling law*, where well-trained neural networks often exhibit power-law scaling relations primarily with the number of neurons, alongside factors such as dataset size and training time (Kaplan et al., 2020; Muennighoff et al., 2024). Despite this, LLMs have inherent limitations in their enclosed reasoning, particularly when addressing complex situations that extend beyond textual boundaries (Schick et al., 2023). To this end, during the inference phase, pioneering studies transform foundational LLMs into versatile *autonomous agents* (Richards, 2023; Shen et al., 2023) by encapsulating external capabilities like context-aware memory (Park et al., 2023), tool use (Qin et al., 2024a), and procedural planning (Zhao et al., 2023). In this context, *multi-agent collaboration*, within an interactive environment, prompts agents to engage in iterative reflection and refinement, explicitly facilitating a process of "slow thinking" (Daniel, 2017; OpenAI, 2024). This paradigm effectively unites the distinct expertise of diverse agents (Qian et al., 2024c), ultimately leading to solutions[1] derived from their dialogues.

Although numerous studies have confirmed that task-oriented multi-agent collaboration, facilitated by interactive behaviors, often surpasses standalone intelligence (Chen et al., 2024d;a), the potential for continuously increasing agents remains largely overlooked—with most research involving fewer than ten agents and only a limited number extending to several dozen (Li et al., 2023a; Park et al., 2023; Zhang et al., 2024a). Inspired by the neural scaling law, a thought-provoking question arises: *how does the continuous addition of collaborative agents impact performance?* Exploring the *collaborative scaling law* is essential for linking performance trends with inference resources, revealing underlying phenomena in agent networking, and promoting the development of scalable and predictable LLM systems. However, technically, effective collaboration should not depend on simple majority voting (Brown et al., 2024; Chen et al., 2024b); instead, it should incorporate strategic mechanisms for scalable networking, cooperative interaction, and progressive decision-making (Hopfield, 1982; Almaatouq et al., 2021; Du et al., 2024a). Toward this end, as depicted in Figure 1, we organize multiple agents into a multi-agent collaboration network (MACNET), upon which their interactive reasoning is topologically orchestrated for autonomous task solving.

- For network construction, agents' topology is constructed as a directed acyclic graph, with each edge managed by a supervisory instructor issuing commands, and each node by a compliant executor providing tailored solutions. This establishes a functional bipartition of labor among agents, promoting role specialization while inherently preventing backflow in information propagation.
- For interactive reasoning, agents interact in a topological order, where each round involves two adjacent agents refining a previous solution, and only the refined solution, rather than the entire dialogue, is propagated to the next rounds. This prevents global broadcasting and suppresses context explosion, thereby enhancing collaboration scalability for much larger networks.

We performed extensive evaluations across different downstream scenarios, employing three types of representative topologies—chain, tree, and graph—further divided into six representative variants. The results show that MACNET surpasses all baselines on average and supports effective collaboration among over a thousand agents. Counterintuitively, collaborating within irregular topologies unexpectedly outperforms that within regular ones. Notably, we reveal a *collaborative scaling law*, indicating that the overall performance exhibits a logistic growth pattern as the process of scaling agents, with collaborative emergence occurring earlier than previous instances of neural emergence. We speculate this may be because scaling agents catalyzes their multidimensional considerations during interactive reflection and refinement, thereby producing more comprehensive solutions. Longer term, we aim for this research to extrapolate the traditional scaling from training to inference, circumventing the need for resource-intensive retraining through inference-time procedural thinking.

2 MULTI-AGENT COLLABORATION NETWORK

To create a scalable environment for effective collaboration, as depicted in Figure 1, we organize multiple agents into a multi-agent collaboration network (MACNET), upon which their interactive reasoning is topologically orchestrated for autonomous task solving.

2.1 NETWORK CONSTRUCTION

Although training-time neuron collaboration has been well-established with Transformer architectures (Vaswani et al., 2017), the suitable architectures for inference-time agent collaboration remain unclear and lack consensus. Toward this end, we draw on the concept of graphs—a data structure that describes entities and their interrelations—and extend from previous efforts to propose a more general topology as a *directed acyclic graph* (DAG) (Nilsson et al., 2020):

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}) \quad \mathcal{V} = \{v_i | i \in I\} \quad \mathcal{E} = \{\langle v_i, v_j \rangle | i, j \in I \wedge i \neq j\} \tag{1}$$

where $\mathcal{V}$ denotes the set of nodes indexed by the index set I, and $\mathcal{E}$ denotes the set of edges, with each edge directed from one node to another and no cycles exist. A graph will orchestrate agent

[1]Solutions can vary from multiple-choice answers to repository-level code or coherent narratives, among many other possibilities.

interactions, akin to social networks where information propagates through directed edges. Intuitively, the acyclic nature prevents information backflow, eliminating the need for additional designs like task-specific cycle-breaking, thereby enhancing generalizability and adaptability across contexts.

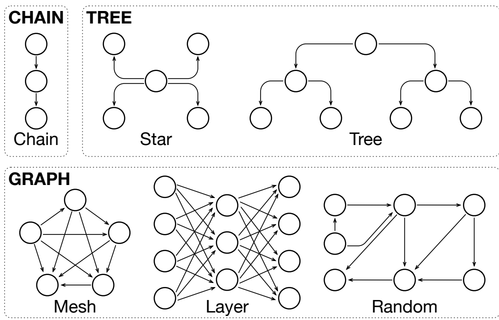


Figure 2: Representative topologies.

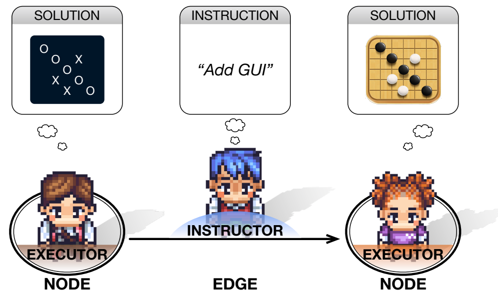


Figure 3: Assign functionally bipartite agents on nodes and edges, respectively.

Given the impracticality of enumerating all possible topologies, we focus on three prevalent types—chain, tree, and graph—further divided into six representative sub-topologies, as depicted in Figure 2. Chain topologies, resembling the waterfall model (Petersen et al., 2009), linearly structuring interactions along agents (Wei et al., 2022b; Hong et al., 2024). Tree topologies enable agents to branch out, interacting in independent directions (Yao et al., 2023; Zhuang et al., 2024); further categorized into "wider" star-shaped and "deeper" tree-shaped topologies. Graph topologies support arbitrary interaction dependencies, with nodes having multiple children and parents, forming either divergent or convergent interactions (Besta et al., 2024a; Chen et al., 2024d; Zhuge et al., 2024; Liu et al., 2023); further classified into fully-connected mesh topologies, MLP-shaped layered topologies, and irregular random topologies. These representative topologies are extensively studied in complex network (Dodds et al., 2003; Newman, 2001; Ma et al., 2024) and procedural reasoning (Zhang et al., 2024b; Yin et al., 2023; Besta et al., 2024b), ensuring a comprehensive coverage of the most widespread and practical topologies in multi-agent networking.

Since a functional bipartition—consisting of supervisory instructors who issue directional instructions and compliant executors who provide tailored solutions—can effectively establish division of labor, activate functional behaviors, and facilitate progressive task-solving (Li et al., 2023a), as depicted in Figure 3, we strategically assign an instructor to each edge and an executor to each node:

$$\boldsymbol{a_i} = \rho(v_i),\ \forall v_i \in \mathcal{V} \quad \boldsymbol{a_{ij}} = \rho(\langle v_i, v_j \rangle),\ \forall \langle v_i, v_j \rangle \in \mathcal{E} \tag{2}$$

where $\rho(x)$ represents the *agentization* operation on an element x, achieved by equipping a foundation model with context-aware memory, external tools, and professional roles; $\boldsymbol{a_i}$ and $\boldsymbol{a_{ij}}$ denote an executor assigned to node v_i and an instructor assigned to edge v_{ij}, respectively.

2.2 Interactive Reasoning

In procedural task-solving, interactive reasoning among agents within a static network requires strategical traversal to establish an orderly interaction criterion (Liu et al., 2024b; Chen et al., 2024e). In a directed acyclic setting, our graph traversal strategy adheres to the principles of *topological ordering* (Kahn, 1962), which ensures that each node is visited only after all its dependencies have been traversed. Formally, for a network $\mathcal{G}$, its topological order is a linear arrangement of agents $\boldsymbol{a_i}$ and $\boldsymbol{a_{ij}}$ such that for every directed edge $\langle v_i, v_j \rangle \in \mathcal{E}$, the ordering satisfies:

$$\forall \langle v_i, v_j \rangle \in \mathcal{E},\ \mathbb{I}(\boldsymbol{a_i}) < \mathbb{I}(\boldsymbol{a_{ij}}) < \mathbb{I}(\boldsymbol{a_j}) \tag{3}$$

where $\mathbb{I}(x)$ denotes the index of agent x in a topological sequence. This arrangement ensures that each node-occupied agent $\boldsymbol{a_i}$ precedes its corresponding edge-occupied agent $\boldsymbol{a_{ij}}$, and $\boldsymbol{a_{ij}}$ precedes $\boldsymbol{a_j}$, thereby ensuring orderly information propogation along the network.

After establishing the global order, as illustrated in Figure 4, we enable each pair of edge-connected adjacent agents to interact for solution refinement, which results in a total assignment of $|\mathcal{V}| + |\mathcal{E}|$ agents and require at least $2 \times |\mathcal{E}|$ interaction rounds. Specifically, within each edge, the interactions

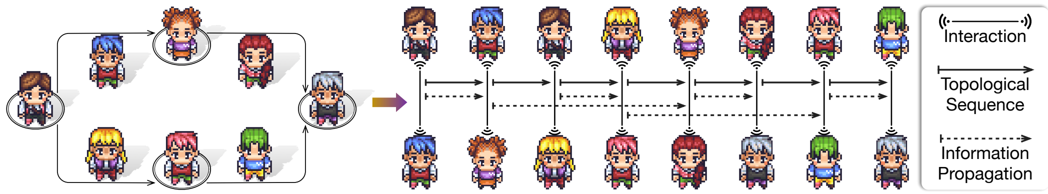


Figure 4: Orchestrating the agents' reasoning process involves a series of dual-agent interactions. The topological order serves as the control flow, while the original connectivity governs the data flow.

between instructors and executors follows a dual-agent multi-turn pattern:

$$\begin{gathered}\tau(\boldsymbol{a_i}, \boldsymbol{a_{ij}}, \boldsymbol{a_j}) = \big(\tau(\boldsymbol{a_i}, \boldsymbol{a_{ij}}), \tau(\boldsymbol{a_{ij}}, \boldsymbol{a_j})\big) \\ \tau(\boldsymbol{a_i}, \boldsymbol{a_{ij}}) = (\boldsymbol{a_i} \rightarrow \boldsymbol{a_{ij}},\ \boldsymbol{a_{ij}} \rightsquigarrow \boldsymbol{a_i})_{\circlearrowleft} \quad \tau(\boldsymbol{a_{ij}}, \boldsymbol{a_j}) = (\boldsymbol{a_{ij}} \rightarrow \boldsymbol{a_j},\ \boldsymbol{a_j} \rightsquigarrow \boldsymbol{a_{ij}})_{\circlearrowleft}\end{gathered} \tag{4}$$

where $\tau(\cdot)$ represents the interaction between agents, $\rightarrow$ signifies an act of requesting, $\rightsquigarrow$ indicates a corresponding reply—within which the instructor provides an instruction and the executor offers a solution, and $\circlearrowleft$ denotes an iterative process. That is, $\boldsymbol{a_i}$ requests feedback, $\boldsymbol{a_{ij}}$ offers reflected suggestions and requests further refinement, and $\boldsymbol{a_j}$ provides a refined solution. Thus, the agents associated with a single edge can engage in iterative reflection and refinement, effectively implementing an refinement of a previous solution (Madaan et al., 2023; Renze & Guven, 2024).[2]

2.3 Memory Control

Note that unrestrained information exchange among agents inevitably leads to *context explosion* (Liu et al., 2024b; Xu et al., 2024), ultimately hindering scalability by limiting support for additional entities. To address this, we adopt both short- and long-term memory to manage the context visibility for each agent (Sumers et al., 2023). *Short-term memory* captures the working memory within each interaction, ensuring context-aware decision-making (Li et al., 2023a). *Long-term memory* maintains context continuity by retaining only the final solution derived from current dialogue, rather than the entire conversational history, ensuring that non-solution contexts (*e.g.,* the detailed analysis process preceding a solution) remain inaccessible[3] to subsequent agents (Qian et al., 2024c). This mechanism ensures that only the solution propagates through the network, which explicitly minimizes context explosion risk while maintaining continuity. Solutions propagate by branching at divergent nodes, or merging at convergent nodes requiring effective aggregation; technically, before refinement, convergent agents integrate the strengths of incoming solutions through hierarchical aggregation (Du et al., 2024b) to yield a "non-linearly" strength-aggregated solution.

Theoretically, in a mesh structure characterized by the highest interaction density, the total token consumption for the sink[4] agent who experiences maximum context pressure, with and without this mechanism, is summarized as follows (refer to the Appendix A for detailed derivations):

$$\begin{aligned}\mathcal{O}(n)_{\text{w/o}} &= t + p + s + (2m-1)(i+s)(n(n-1)/2 + 2(n-2)) \overset{n \gg 1}{\approx} Cn^2 \propto n^2 \\ \mathcal{O}(n)_{\text{w/}} &= t + p + s + m(i+s)((n-1) + 2(n-2)) \overset{n \gg 1}{\approx} \bar{C}n \propto n \\ \text{where} \quad & C \equiv (2m-1)(i+s)/2 \quad \bar{C} \equiv 3m(i+s)\end{aligned} \tag{5}$$

where n is the network scale (*i.e.,* $|\mathcal{V}|$), t the task length, p the profile length, i the average instruction length, s the average solution length, and m the maximum interaction rounds between adjacent agents. This token complexity analysis implies that, without memory control, context length grows with n^2, causing squared increases in time and cost as the network scales.[5] Conversely, our mechanism

[2]Note that although the interaction order is unfolded as a sequence for visualization purposes only, certain sub-topologies (*e.g.,* star) inherently support parallel processing.

[3]Inaccessibility doesn't mean abandonment; when agents incorporate previous contexts into a solution, these contexts are implicitly embedded and carried forward with the solution.

[4]The "sink agent" refers to the agent assigned to the sink node. In a multi-sink structure, a final sink node is automatically appended to form a structure with only one sink.

[5]Empirical evidence shows that in mesh topologies with $n \geq 7$ within a 16k window, the absence of memory control almost invariably leads to context explosion issues, causing the entire reasoning process to fail.

decouples context length from quadratic to linear growth, effectively suppressing context explosion and enabling better scalability for larger networks.

3 EVALUATION

Baselines We select a diverse set of representative methods to facilitate a comprehensive multidimensional comparison:

- COT (Wei et al., 2022b) is a technically general and empirically powerful approach that endows LLMs with the ability to generate a coherent series of intermediate reasoning steps, naturally leading to the final solution through process-aware thoughtful thinking.
- AUTOGPT (Richards, 2023) is a versatile agent that employs multi-step planning and tool-augmented reasoning to decompose complex tasks into chained subtasks and leverages external tools within an environment-feedback cycle to progressively develop effective solutions.
- GPTSWARM (Zhuge et al., 2024) formalizes a swarm of autonomous agents as computational graphs, with nodes as manually-customized functions and edges facilitating information flow, adaptively optimizing node prompts and modifying graph connectivity during collective reasoning.
- AGENTVERSE (Chen et al., 2024d) dynamically assembles and coordinates a team of expert agents in chained or hierarchical structures, employing multi-agent linguistic interaction to autonomously reflect and refine solutions while displaying emergent social behaviors.

Datasets and Metrics We adopt publicly available and logically challenging benchmarks to evaluate performance across heterogeneous downstream scenarios.

- MMLU (Hendrycks et al., 2021) provides a comprehensive set of logical reasoning assessments across diverse subjects and difficulties, utilizing multiple-option questions to measure general world knowledge and logical inference capabilities. We assess the quality of generated solutions via *accuracy*, which reflects the correctness of responses to multiple-choice questions.
- HumanEval (Chen et al., 2021), a widely recognized benchmark for function-level code generation, designed for measuring basic programming skills. We assess via *pass@k*, which reflects function correctness across multiple standard test cases.
- SRDD (Qian et al., 2024c) integrates complex textual software requirements from major real-world application platforms, tailored for repository-level software development, involving requirement comprehension, system design, code generation and testing. We assess using the official comprehensive metric encompassing completeness, executability, and consistency.
- CommonGen-Hard (Madaan et al., 2023) tests the ability to generate coherent sentences with discrete concepts, assessing contextual understanding, commonsense reasoning, and creative writing skills. We assess using a comprehensive metric that integrates crucial factors including grammar, fluency, context relevance, and logic consistency (Li et al., 2018).

Implementation Details We construct non-deterministic topologies such as trees and graphs utilizing fundamental structures, including binary trees, layered structures balanced in both width and depth, and random structures crafted by removing edges from a mesh while maintaining connectivity. By default, we employ a topology consisting of approximately four nodes, aligning with multi-agent baselines. In interactive reasoning, GPT-4 is utilized to generate diverse role-specific profiles and outline the available tools for agentization, which are then randomly sampled and assigned to networked agents. GPT-3.5 is employed for interactive reasoning due to its optimal balance of efficacy and efficiency, with each iterative interaction limited to three exchange rounds. To ensure fairness, all baselines are configured with identical settings.

3.1 DOES OUR METHOD LEAD TO IMPROVED PERFORMANCE?

We employ the simplest topology—chain—as the default setting for comparative analysis. As demonstrated in Table 1, the chain-structured method consistently surpasses all baselines across most metrics, showing a significant margin of improvement. The primary advantage of MACNET-CHAIN, over a single agent who provides solutions directly, lies in its facilitation of a procedural thinking in

Method	Paradigm	MMLU	HumanEval	SRDD	CommonGen	Quality
CoT		0.3544†	0.6098†	0.7222†	0.6165†	0.5757†
AutoGPT		0.4485†	0.4809†	0.7353†	0.5972	0.5655†
GPTSwarm		0.2368†	0.4969†	0.7096†	0.6222†	0.5163†
AgentVerse		0.2977†	**0.7256**†	0.7587†	0.5399†	0.5805
MacNet-Chain		0.6632	0.3720	**0.8056**	0.5903	0.6078
MacNet-Star		0.4456†	0.5549†	0.7679†	0.7382†	0.6267
MacNet-Tree		0.3421†	0.4878†	0.8044	**0.7718**†	0.6015
MacNet-Mesh		0.6825	0.5122†	0.7792†	0.5525†	0.6316†
MacNet-Layer		0.2780†	0.4939†	0.7623†	0.7176†	0.5629†
MacNet-Random		**0.6877**	0.5244†	0.8054	0.5912	**0.6522**†

Table 1: The overall performance of LLM-driven methods across various datasets, including both single-agent () and multi-agent () paradigms. Quality represents the average performance over all tasks. For each dataset, the highest scores are highlighted in bold, while the second-highest scores are underlined. A dagger (†) denotes statistically significant differences ($p \leq 0.05$) between a method and our chain-structured setting.

which solutions are continually reflected and refined. This process effectively mitigates previous inaccuracies or unexpected hallucinations, aligning with previous findings (Cohen et al., 2023; Du et al., 2024a; Qian et al., 2024b). Moreover, we observe that CoT exhibits strong performance on certain datasets, which is largely because the underlying knowledge of widely-researched benchmarks is already embedded in foundational models, giving single agents a notable capability in these relatively "simple" tasks. While GPTSwarm self-organizes agents through dynamic optimization of nodes and edges, it unfortunately necessitates extensive task-specific customization for all nodes and edges, complicating usage and thus hindering seamless generalization to heterogeneous downstream tasks. Given the growing need for highly performant and automatic real-world systems, it is impractical to expect that all preparatory knowledge can be fully pre-encoded in foundation models, nor can specific adaptations be pre-made for all unforeseen complex tasks. Fortunately, MacNet bridges this gap by automatically generating various networks through simple hyperparameters (*e.g.,* topology type and scale), enabling agents to engage in cooperative interactions without needing specific adjustments, which represents a promising pathway to achieving both autonomy and generalizability. Furthermore, we simulate a regression to graph-of-thought reasoning (Besta et al., 2024a) with a simplified agent by ablating agents' profiles, which led to an average performance drop of 3.67% across all topologies. This result underscores the effectiveness of collective intelligence over singular-aspect reasoning, as the latter represents a variant of dimensionality reduction within multi-agent environments, inevitably blocking its potential to extrapolate potential opportunities.

3.2 How Do Different Topologies Perform Against Each Other?

To gain a deeper understanding of the impact on organizational structures within multi-agent collaboration, we examine MacNet's topologies across six representative topologies. The analysis focuses on three key perspectives: density, shape, and direction.

Density Perspective Table 1 illustrates that different types of topologies vary significantly in effectiveness for specific tasks; no single topology consistently excels across all tasks. For instance, a chain topology is more suitable for software development, while a tree topology is ideal for creative writing. This phenomenon may arise from the inherent suitability of software engineering to a linear process, which is accomplished through sequential steps such as analysis, coding, review, and testing; in contrast, tasks requiring high creativity necessitate more divergent structures to foster agent interactions from various aspects. Additionally, higher interaction density, associated with edge density (see Figure 5), correlates with improved average performance across the three primary topological types. Specifically, the densely connected mesh topology outperforms the moderately dense tree topology, which in turn outperforms the sparsely connected chain topology. This can be

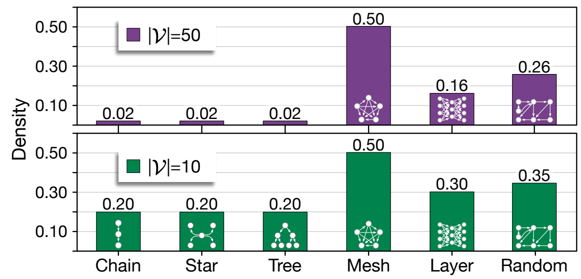


Figure 5: Density of different topologies at different scales.

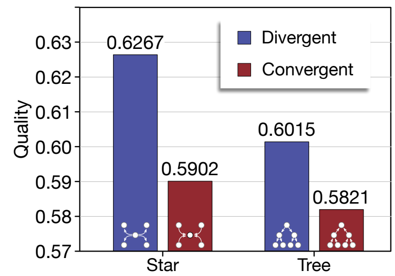


Figure 6: Comparison between topologies and their reversed counterparts.

attributed to the fact that increased density natually prolongs the reasoning process among collective agents, potentially enhancing opportunities for optimizing solutions from various aspects.

Shape Perspective Despite the intuitive appeal of densest interactions (*i.e.,* mesh), they do not always yield optimal performance. In contrast, irregular topologies often demonstrate statistically significant advantages. We hypothesize that this phenomenon is because overly dense interactions can overwhelm agents with information overload, impeding effective reflection and refinement. Conversely, network randomization frequently induces small-world properties (Watts & Strogatz, 1998), characterized by a shorter *average path length*[6] or a higher *clustering coefficient*[7]. These random edge connections, akin to residual connections, can link "unacquainted" agents via direct shortcuts, transforming them into "acquaintances" and implicitly reducing the average path length, which naturally decreases the likelihood of long-distance solution invisibility. This phenomenon, seemingly counterintuitive when compared to well-established regular organizational structures in the real world, suggests that collaboration patterns in an agent's world need not precisely mirror those in human society. Additionally, random topologies consume approximately 51.92% less time than mesh topologies, striking an optimal balance between reduced density and enhanced efficiency, thus serving as a more practical choice. It has also been noticed that, with the same density, star-shaped topologies that are "wider" tend to perform better than "deeper" tree-shaped ones. This is primarily due to the memory control mechanism; while it efficiently manages the spread of overly lengthy contexts across the network, it may cause deeper topologies to lose track of distant agents, occasionally resulting in solution version rollbacks (Qian et al., 2024a). This points to an empirical search strategy that manages network scale and clustering coefficients, whether through automated searching or manual design, to find an optimal balance between effectiveness and efficiency. Delving deeper, an in-depth inductive bias analysis reveals that in closed-domain scenarios (*e.g.,* logical choices), a chain structure significantly aids in facilitating step-by-step reasoning. Conversely, a proliferation of parallel branches (*e.g.,* stars) can lead to convoluted brainstorming, which may not always be advantageous. In open-domain scenarios, topologies characterized by more convergent nodes are shown to revise solutions more frequently and produce longer solutions[8]. This occurs because more convergent nodes, with increased input diversity, increase the likelihood of refining solutions, benefiting length-sensitive metrics as longer solutions are more likely to meet rich requirements. Ultimately, no task is confined to a particular topology; the optimal configuration should be chosen based on the openness of scenarios, available computing resources, and associated reasoning costs.

Direction Perspective Beyond density and shape perspectives, the inherent asymmetry in certain topologies—where reversing the edges results in a topologically distinct configuration—has interested us in exploring the effects of reversed topologies. As shown in Figure 6, merely reversing the directions of specific topologies can lead to significant performance degradation. Typically, divergent topologies, characterized by having more child nodes than parent nodes, substantially outperform their convergent counterparts. Intuitively, solution propagation diverges smoothly, enabling each

[6]Average path length (Albert & Barabasi, 2002) is the average number of steps along the shortest paths for all possible pairs of network nodes, which is a measure of the efficiency of information transport on a network.

[7]The clustering coefficient measures the connectivity density among a node's neighbors (Strogatz, 2001).

[8]The layer topologies exhibit a 92.16% modification probability and an average solution length of 586.57, compared to 68.48% and 308.26 for chain topologies

agent to discuss solutions from varied aspects. In contrast, aggregating multiple solutions at a convergent node is more challenging, highlighting the complexity of integrating diverse aspects into a cohesive solution. Therefore, to minimize potential degradation during solution aggregation, it is recommended to employ topologies that maximize divergence while minimizing convergence.

3.3 Could A Collaborative Scaling Law Be Observed?

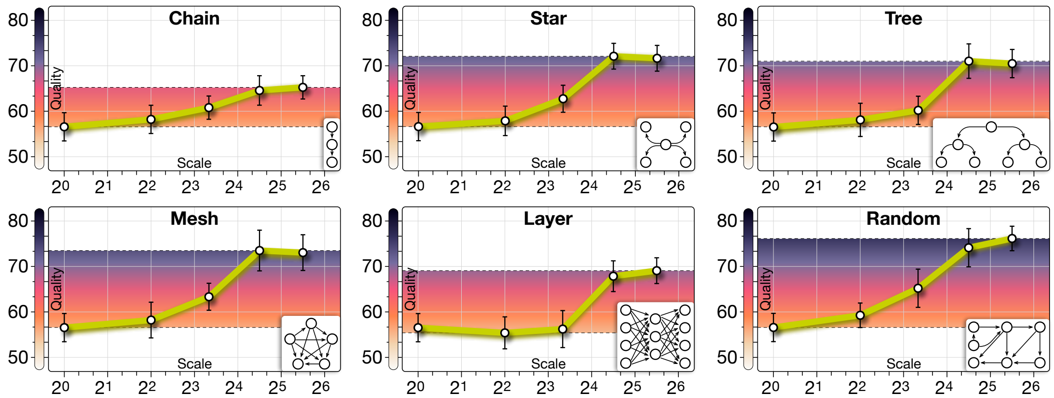


Figure 7: Scaling performance of multi-agent collaboration under different topologies. Quality represents the average performance over all tasks.

Trend Perspective Recall the neural scaling law, which posits that increasing neurons leads to an continual performance improvement (Kaplan et al., 2020). To investigate the *collaborative scaling law*, which excavates the relationship between agent scale and performance, we initiated an attempt by exponentially increasing the number of nodes ($|\mathcal{V}|$) from 2^0 (regressing to a single-agent variant) to 2^6 (equating to over a thousand agents in a mesh network). As depicted in Figure 7, scaling our networks initially grows slowly in the quality of solutions generated by various multi-agent systems, then leads to a rapid improvement before reaching a saturation point. This pattern resembles a sigmoid-variant function:

$$f(|\mathcal{V}|) = \frac{\gamma}{1 + e^{-\beta(\log|\mathcal{V}| - \alpha)}} + \delta \tag{6}$$

where $\{\alpha, \beta, \gamma, \delta\}$ are real numbers specific to a particular topology. Roughly speaking, a node magnitude of 2^4 appears to be a reasonable choice. However, considering the efficiency of sparse topologies and the superior performance of dense ones, we advocate balancing shape and scale through multidimensional trade-offs when applying this trend to various downstream applications. This finding suggests that many existing agent systems may be operating below their full potential, which underscores a promising path for enhancing performance by increasing the number of agents, provided they collaborate effectively, rather than solely focusing on scaling foundational models.[9]

Besides, the validation of baseline scaling reveals that equalizing the number of LLM calls—whether through majority voting in closed-domain tasks (Chen et al., 2024b) or best-of-N in open-domain tasks (Sessa et al., 2024)—consistently highlights a lack of effective scalability across all baselines. Majority voting enhances performance by merely 0.9%, even when augmented with CoT or AutoGPT, plateauing at approximately eight agents. AgentVerse implicitly reduces to a star topology and frequently encounters context explosion issues when scaling beyond thirty agents, thus hindering scalability. The energy-intensive setup of GPTSwarm necessitates manual, task-specific structuring and prompting, which restricts both multitasking capabilities and overall scalability.

Timing Perspective The neural scaling law requires models with at least a billion parameters and over 10^{22} training FLOPs to show emergent trends (Schaeffer et al., 2024). In contrast, collaborative emergence in MacNet manifests at much smaller scales, with most topologies reaching performance

[9]Looking further, this fitting only reflects a general pattern from the perspective of network scales; future research should aim for a more precise characterization by incorporating additional factors like profiles, tools and communication protocols, or social routing.

saturation with approximately a hundred agents. The fundamental reason is that neuron coordination (during training) relies on numeric matrix operations, requiring all neurons to precisely and simultaneously learn from scratch to assimilate extensive world knowledge. Conversely, individual agents (during inference) already possess certain knowledge from the foundational models, and their coordination through interdependent interactions utilizes existing reasoning skills to disseminate knowledge from diverse aspects; the most critical aspects for solution refinement in agents' interactions typically do not require such a large scale to be thoroughly reflected and refined. Thus, alongside neuron collaboration, agent collaboration may serve as a "shortcut" to enhance intelligence levels, especially when large-scale retraining resources such as data and hardware are constrained.

3.4 What Factors Might Contribute to Collaborative Emergence?

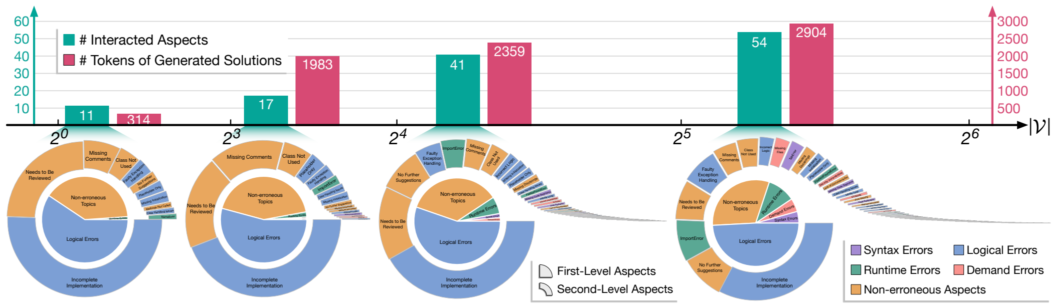


Figure 8: The number and distribution of aspects in agent interactions, along with the length of solutions. The pie chart features primary aspects in the inner circle and secondary aspects in the outer circle, with a long-tail layout to visualize tail aspects. Zoom in for more detailed information.

To delve deeper into the underlying mechanisms, we selected the moderately-dense layer typology employed in software development, which serves as a representative case, with similar phenomena consistently occurring in other topologies and scenarios. Specifically, we classified the aspects discussed in agents' interactions into five main categories (Oh & Oh, 2022; Kohn, 2019): four levels of errors (syntax, runtime, logic, and unmet requirements) and a non-error category; each category contains multiple subcategories. Figure 8 displays the total number of interaction aspects, along with their detailed distribution. Within smaller topologies ($2^0 \leq |\mathcal{V}| \leq 2^3$), the limited interaction density confines aspects to approximately a dozen secondary aspects. However, as the network expands ($2^4 \leq |\mathcal{V}| \leq 2^6$), the interaction density increases quadratically, resulting in a sudden increase to dozens of aspects, followed by a more gradual rise. This progression closely parallels the trend observed in emergent capabilities, which may partially attribute the emergence to the sharp rise in detailed interacted aspects among agents. This phenomenon occurs because the token distribution from underlying models typically follows a long-tail pattern, necessitating larger-scale sampling to likely capture these tail tokens. Consequently, this encourages the emergence of more infrequent "tail aspects", allowing the collaborative process to extend beyond the most common aspects. Theoretically, the probability of a long-tail token t appearing at least once in n samples is:

$$p^n(t) = 1 - (1 - p(t))^n \propto 1 - (1 - 1/r(t))^{|\mathcal{V}|^2} \qquad \lim_{|\mathcal{V}| \to \infty} p^n(t) = \lim_{n \to \infty} p^n(t) = 1 \tag{7}$$

where $p(t) \propto 1/r(t)$ represents a standard Zipf's law characterizing a long-tailed distribution (Newman, 2005); the sampling size n is proportional to the interaction density, *i.e.*, $n \propto |\mathcal{V}|^2$. It can be inferred that increasing the network size significantly enhances the probability of tail token occurrences, gradually approaching an asymptote. This probability becomes an inevitable event once the sample size is sufficiently large. Statistically, when an instructor suggests a particular aspect, there is a 93.10% statistical likelihood that an executor will implement the recommended refinement rather than disregard it. The scaling up enables instructors to pinpoint finer issues within solutions, guiding executors to initiate corresponding refinements. Consequently, each round of dialogue in the collaborative process refines solutions from different aspects, naturally elevating the probability of producing more nuanced solutions (Liang et al., 2024; Du et al., 2024a; Cohen et al., 2023).

In response to multidimensional considerations, scaling agents accordingly prolongs the overall length of solutions. For instance, the token length increased by 7.51 times when scaling from 2^0 to

2^4. This characteristic, over small-scale networks, facilitates the integration of detailed requirements, performance optimization, and other advanced factors, potentially encompassing abilities that shorter solutions cannot. This is mainly due to the graph's naturally divergent and convergent topologies, which enable solutions to porpagate for strength-aggregated refinement. Therefore, unlike majority voting, this paradigm fosters interdependent interaction and length-extended regeneration among diversified solutions, thereby producing more comprehensive solutions (Appendix E for case study).

4 RELATED WORK

Large Language Models Trained on vast datasets through next token prediction (Vaswani et al., 2017) and capable of manipulating billions of parameters (Muennighoff et al., 2024), LLMs have become pivotal in natural language processing due to their seamless integration of extensive knowledge (Brown et al., 2020; Bubeck et al., 2023; Radford et al., 2019; Touvron et al., 2023; Wei et al., 2022a; Shanahan et al., 2023; Chen et al., 2021; Brants et al., 2007; Chen et al., 2021; Ouyang et al., 2022; Yang et al., 2024; Qin et al., 2024b). Central to this breakthrough is the neural scaling law, which posits that loss descends as a power law with model size, dataset size, and the amount of compute used for training (Kaplan et al., 2020; Smith et al., 2022; Ruan et al., 2024). The principle underscores that scaling up language models can lead to emergent abilities—where performance experiences a sudden leap as the model scales (Wei et al., 2022a; Schaeffer et al., 2024).

Autonomous Agents Despite these advancements, LLMs possess inherent limitations in enclosed reasoning, driving further research to integrate advanced capabilities such as context-aware memory (Park et al., 2023; Hua et al., 2023), tool use (Schick et al., 2023; Qin et al., 2024a), procedural planning (Wang et al., 2023a; Zelikman et al., 2024), and role playing (Chan et al., 2024; Wang et al., 2024c; Liu et al., 2024a), thereby transforming fundamental LLMs into versatile autonomous agents (Richards, 2023; Shinn et al., 2024; Zhao et al., 2024; Lin et al., 2023; Mei et al., 2024; Chu et al., 2024). Along this line, multi-agent collaboration has proven beneficial in uniting the expertise of diverse agents for autonomous task-solving (Khan et al., 2024; Liang et al., 2024; Qian et al., 2024c; Wang et al., 2024b;a; Zhou et al., 2024; Talebirad & Nadiri, 2023; Chen et al., 2024c; Li et al., 2023b), which has widely propelled progress across various domains such as software development (Hong et al., 2024; Qian et al., 2024a), game playing (Vinyals et al., 2019), personalized recommendation (Wang et al., 2023b; Zhang et al., 2023), medical treatment (Tang et al., 2023; Li et al., 2024a), financial marketing (Gao et al., 2024; Li et al., 2024c), educational teaching (Zhang et al., 2024c; Yu et al., 2024), scientific research (Zeng et al., 2024; Baek et al., 2024; Ghafarollahi & Buehler, 2024) and embodied control (Guo et al., 2024; Chen et al., 2024f; Mandi et al., 2023). Technically, in contrast to straightforward majority voting where individuals act independently (Chen et al., 2024b), collective emergence (Woolley et al., 2010; Hopfield, 1982; Watts & Strogatz, 1998) posits that effective collaboration should evolve into an integrated system that promotes interdependent interactions and thoughtful decision-making (Li et al., 2024b; Piatti et al., 2024). As such, recent studies differentiate agents into distinct expertise and encourage task-oriented interactions, forming a chained workflow to sequentially reach final solutions (Qian et al., 2024c). Subsequent research seeks to organize expert agents in a tree structure for hierarchical information propagation (Chen et al., 2024d) or in a graph with predefined node and edge functions (Zhuge et al., 2024).

5 CONCLUSION

This study explores the impact of scaling multi-agent collaboration by introducing MACNET, a scalable framework that utilizes graphs to organize agents and orchestrate their reasoning for autonomous task solving. Extensive evaluations reveal that it effectively supports collaboration among over a thousand agents, with irregular topologies outperforming regular ones. We also identify a *collaborative scaling law*—the overall performance follows a logistic growth pattern as agents scale, with collaborative emergence occurring earlier than previously observed neural emergence. We speculate this may be because scaling agents catalyzes their multidimensional considerations during interactive reflection and refinement, thereby producing more comprehensive solutions. However, our research also indicates that there are limits on the scaling horizon. By extrapolating traditional scaling from training to inference, we posit that agent collaboration could serve as a "shortcut" to bypass the need for resource-intensive retraining by employing inference-time procedural thinking.

References

Reka Albert and Albert-Laszlo Barabasi. Statistical Mechanics of Complex Networks. In *Reviews of Modern Physics*, 2002. URL https://arxiv.org/abs/cond-mat/0106096.

Abdullah Almaatouq, Mohammed Alsobay, Ming Yin, and Duncan J. Watts. Task Complexity Moderates Group Synergy. In *National Academy Of Sciences (PNAS)*, 2021. URL https://www.pnas.org/doi/full/10.1073/pnas.2101062118.

Jinheon Baek, Sujay Kumar Jauhar, Silviu Cucerzan, and Sung Ju Hwang. ResearchAgent: Iterative Research Idea Generation over Scientific Literature with Large Language Models. In *arXiv preprint arXiv:2404.07738*, 2024. URL https://arxiv.org/pdf/2404.07738.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of Thoughts: Solving Elaborate Problems with Large Language Models. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2024a. URL https://arxiv.org/pdf/2308.09687.

Maciej Besta, Florim Memedi, Zhenyu Zhang, Robert Gerstenberger, Guangyuan Piao, Nils Blach, Piotr Nyczyk, Marcin Copik, Grzegorz Kwaśniewski, Jürgen Müller, Lukas Gianinazzi, Ales Kubicek, Hubert Niewiadomski, Aidan O'Mahony, Onur Mutlu, and Torsten Hoefler. Demystifying Chains, Trees, and Graphs of Thoughts. In *arXiv preprint arXiv:2401.14295*, 2024b. URL https://arxiv.org/pdf/2401.14295.

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. Large Language Models in Machine Translation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2007. URL https://aclanthology.org/D07-1090/.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large Language Monkeys: Scaling Inference Compute with Repeated Sampling. In *arXiv preprint arXiv:2407.21787*, 2024. URL https://arxiv.org/abs/2407.21787.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of Artificial General Intelligence: Early Experiments with GPT-4. In *arXiv preprint arXiv:2303.12712*, 2023. URL https://doi.org/10.48550/arXiv.2303.12712.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. ChatEval: Towards Better LLM-based Evaluators through Multi-agent Debate. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://iclr.cc/virtual/2024/poster/19065.

Justin Chen, Swarnadeep Saha, and Mohit Bansal. ReConcile: Round-Table Conference Improves Reasoning via Consensus among Diverse LLMs. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024a. URL https://aclanthology.org/2024.acl-long.381/.

Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei Zaharia, and James Zou. Are More LLM Calls All You Need? Towards Scaling Laws of Compound Inference Systems. In *arXiv preprint arXiv:2403.02419*, 2024b. URL https://arxiv.org/pdf/2403.02419.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating Large Language Models Trained on Code. In *arXiv preprint arXiv:2107.03374*, 2021. URL https://arxiv.org/pdf/2107.03374.

Pei Chen, Shuai Zhang, and Boran Han. CoMM: Collaborative Multi-Agent, Multi-Reasoning-Path Prompting for Complex Problem Solving. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2024c. URL https://aclanthology.org/2024.findings-naacl.112/.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, et al. AgentVerse: Facilitating Multi-agent Collaboration and Exploring Emergent Behaviors in Agents. In *International Conference on Learning Representations (ICLR)*, 2024d. URL https://iclr.cc/virtual/2024/poster/19109.

Weize Chen, Ziming You, Ran Li, Yitong Guan, Chen Qian, Chenyang Zhao, Cheng Yang, Ruobing Xie, Zhiyuan Liu, and Maosong Sun. Internet of Agents: Weaving a Web of Heterogeneous Agents for Collaborative Intelligence. In *arXiv preprint arXiv:2407.07061*, 2024e. URL https://arxiv.org/pdf/2407.07061.

Yongchao Chen, Jacob Arkin, Yang Zhang, Nicholas Roy, and Chuchu Fan. Scalable Multi-Robot Collaboration with Large Language Models: Centralized or Decentralized Systems? In *arXiv preprint arXiv:2309.15943*, 2024f. URL https://arxiv.org/pdf/2309.15943.

Zhixuan Chu, Yan Wang, Feng Zhu, Lu Yu, Longfei Li, and Jinjie Gu. Professional Agents – Evolving Large Language Models into Autonomous Experts with Human-Level Competencies. In *arXiv preprint arXiv:2402.03628*, 2024. URL https://arxiv.org/pdf/2402.03628.

Roi Cohen, May Hamri, Mor Geva, and Amir Globerson. LM vs LM: Detecting Factual Errors via Cross Examination. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2023. URL https://aclanthology.org/2023.emnlp-main.778/.

Kahneman Daniel. Thinking, Fast and Slow. In *Farrar, Straus and Giroux*, 2017. URL https://www.pdcnet.org//collection/fshow?id=inquiryct_2012_0027_0002_0054_0057&pdfname=inquiryct_2012_0027_0002_0055_0058.pdf&file_type=pdf.

Peter Sheridan Dodds, Duncan J. Watts, and Charles F. Sabel. Information Exchange and the Robustness of Organizational Networks. In *National Academy Of Sciences (PNAS)*, 2003. URL https://www.pnas.org/doi/abs/10.1073/pnas.1534702100.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving Factuality and Reasoning in Language Models through Multiagent Debate. In *International Conference on Machine Learning (ICML)*, 2024a. URL https://openreview.net/pdf?id=zj7YuTE4t8.

Zhuoyun Du, Chen Qian, Wei Liu, Zihao Xie, Yifei Wang, Yufan Dang, Weize Chen, and Cheng Yang. Multi-Agent Software Development through Cross-Team Collaboration. In *arXiv preprint arXiv:2406.08979*, 2024b. URL https://arxiv.org/pdf/2406.08979.

Shen Gao, Yuntao Wen, Minghang Zhu, Jianing Wei, Yuhan Cheng, Qunzi Zhang, and Shuo Shang. Simulating Financial Market via Large Language Model based Agents. In *arXiv preprint arXiv:2406.19966*, 2024. URL https://arxiv.org/pdf/2406.19966.

Alireza Ghafarollahi and Markus J. Buehler. SciAgents: Automating Scientific Discovery through Multi-Agent Intelligent Graph Reasoning. In *arXiv preprint arXiv:2409.05556*, 2024. URL https://arxiv.org/pdf/2409.05556.

Xudong Guo, Kaixuan Huang, Jiale Liu, Wenhui Fan, Natalia Vélez, Qingyun Wu, Huazheng Wang, Thomas L. Griffiths, and Mengdi Wang. Embodied LLM Agents Learn to Cooperate in Organized Teams. In *arXiv preprint arXiv:2403.12482*, 2024. URL https://arxiv.org/pdf/2403.12482.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Xiaodong Song, and Jacob Steinhardt. Measuring Massive Multitask Language Understanding. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://api.semanticscholar.org/CorpusID:221516475.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, 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 *International Conference on Learning Representations (ICLR)*, 2024. URL `https://iclr.cc/virtual/2024/poster/18491`.

J J Hopfield. Neural Networks and Physical Systems with Emergent Collective Computational Abilities. In *National Academy Of Sciences (PNAS)*, 1982. URL `https://doi.org/10.1073/pnas.79.8.2554`.

Wenyue Hua, Lizhou Fan, Lingyao Li, Kai Mei, Jianchao Ji, Yingqiang Ge, Libby Hemphill, and Yongfeng Zhang. War and Peace (WarAgent): Large Language Model-based Multi-Agent Simulation of World Wars. In *arXiv preprint arXiv:2311.17227*, 2023. URL `https://arxiv.org/pdf/2311.17227`.

A. B. Kahn. Topological Sorting of Large Networks. In *Communications of the ACM*, 1962. URL `https://dl.acm.org/doi/10.1145/368996.369025`.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling Laws for Neural Language Models. In *arXiv preprint arXiv:2001.08361*, 2020. URL `https://doi.org/10.48550/arXiv.2001.08361`.

Akbir Khan, John Hughes, Dan Valentine, Laura Ruis, Kshitij Sachan, Ansh Radhakrishnan, Edward Grefenstette, Samuel R. Bowman, Tim Rocktäschel, and Ethan Perez. Debating with More Persuasive LLMs Leads to More Truthful Answers. In *International Conference on Machine Learning (ICML)*, 2024. URL `https://icml.cc/virtual/2024/poster/33360`.

Tobias Kohn. The Error Behind The Message: Finding the Cause of Error Messages in Python. In *ACM Technical Symposium on Computer Science Education (SIGCSE)*, 2019. URL `https://doi.org/10.1145/3287324.3287381`.

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 *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a. URL `https://arxiv.org/abs/2303.17760`.

Junkai Li, Siyu Wang, Meng Zhang, Weitao Li, Yunghwei Lai, Xinhui Kang, Weizhi Ma, and Yang Liu. Agent Hospital: A Simulacrum of Hospital with Evolvable Medical Agents. In *arXiv preprint arXiv:2405.02957*, 2024a. URL `https://arxiv.org/pdf/2405.02957`.

Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. More Agents is All You Need. In *arXiv preprint arXiv:2402.05120*, 2024b. URL `https://arxiv.org/pdf/2402.05120`.

Nian Li, Chen Gao, Mingyu Li, Yong Li, and Qingmin Liao. EconAgent: Large Language Model-Empowered Agents for Simulating Macroeconomic Activities. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024c. URL `https://aclanthology.org/2024.acl-long.829/`.

Yuan Li, Yixuan Zhang, and Lichao Sun. MetaAgents: Simulating Interactions of Human Behaviors for LLM-based Task-oriented Coordination via Collaborative Generative Agents. In *arXiv preprint arXiv:2310.06500*, 2023b. URL `https://arxiv.org/pdf/2310.06500`.

Zhongyang Li, Xiao Ding, and Ting Liu. Generating Reasonable and Diversified Story Ending using Sequence to Sequence Model with Adversarial Training. In *International Conference on Computational Linguistics (COLING)*, 2018. URL `https://aclanthology.org/C18-1088/`.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Rui Wang, Yujiu Yang, Zhaopeng Tu, and Shuming Shi. Encouraging Divergent Thinking in Large Language Models through Multi-Agent Debate. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2024. URL `https://arxiv.org/pdf/2305.19118`.

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. SwiftSage: A Generative Agent with Fast and Slow Thinking for Complex Interactive Tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/pdf/2305.17390.

Ruibo Liu, Ruixin Yang, Chenyan Jia, Ge Zhang, Diyi Yang, and Soroush Vosoughi. Training Socially Aligned Language Models on Simulated Social Interactions. In *International Conference on Learning Representations (ICLR)*, 2024a. URL https://arxiv.org/pdf/2305.16960.

Wei Liu, Chenxi Wang, Yifei Wang, Zihao Xie, Rennai Qiu, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, and Chen Qian. Autonomous Agents for Collaborative Task under Information Asymmetry. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024b. URL https://arxiv.org/pdf/2406.14928.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. Dynamic LLM-Agent Network: An LLM-agent Collaboration Framework with Agent Team Optimization. In *arXiv preprint arXiv:2310.02170*, 2023. URL https://arxiv.org/pdf/2310.02170.

Chengdong Ma, Aming Li, Yali Du, Hao Dong, and Yaodong Yang. Efficient and Scalable Reinforcement Learning for Large-scale Network Control. In *Nature Machine Intelligence (NMI)*, 2024. URL https://doi.org/10.1038/s42256-024-00879-7.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative Refinement with Self-Feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf.

Zhao Mandi, Shreeya Jain, and Shuran Song. RoCo: Dialectic Multi-Robot Collaboration with Large Language Models. In *arXiv preprint arXiv:2307.04738*, 2023. URL https://arxiv.org/pdf/2307.04738.

Kai Mei, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. AIOS: LLM Agent Operating System. In *arXiv preprint arXiv:2403.16971*, 2024. URL https://arxiv.org/pdf/2403.16971.

Niklas Muennighoff, Alexander Rush, Boaz Barak, Teven Le Scao, Nouamane Tazi, Aleksandra Piktus, Sampo Pyysalo, Thomas Wolf, and Colin Raffel. Scaling Data-Constrained Language Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/9d89448b63ce1e2e8dc7af72c984c196-Abstract-Conference.html.

M. E. J. Newman. The Structure of Scientific Collaboration Networks. In *National Academy Of Sciences (PNAS)*, 2001. URL https://www.pnas.org/doi/full/10.1073/pnas.98.2.404.

MEJ Newman. Power laws, Pareto distributions and Zipf's law. In *Contemporary Physics*, 2005. URL https://www.tandfonline.com/doi/abs/10.1080/00107510500052444.

Anton Nilsson, Carl Bonander, Ulf Strömberg, and Jonas Björk. A Directed Acyclic Graph for Interactions. In *International Journal of Epidemiology*, 2020. URL https://doi.org/10.1093/ije/dyaa211.

Wonseok Oh and Hakjoo Oh. PyTER: Effective Program Repair for Python Type Errors. In *ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2022. URL https://dl.acm.org/doi/10.1145/3540250.3549130.

OpenAI. Learning to Reason with LLMs. In *https://openai.com/index/learning-to-reason-with-llms*, 2024. URL https://openai.com/index/learning-to-reason-with-llms/.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training Language Models to Follow Instructions with Human Feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative Agents: Interactive Simulacra of Human Behavior. In *Annual ACM Symposium on User Interface Software and Technology (UIST)*, 2023. URL https://doi.org/10.1145/3586183.3606763.

Kai Petersen, Claes Wohlin, and Dejan Baca. The Waterfall Model in Large-Scale Development. In *Product-Focused Software Process Improvement*, 2009. URL https://doi.org/10.1007/978-3-642-02152-7_29.

Giorgio Piatti, Zhijing Jin, Max Kleiman-Weiner, Bernhard Schölkopf, Mrinmaya Sachan, and Rada Mihalcea. Cooperate or Collapse: Emergence of Sustainability Behaviors in a Society of LLM Agents. In *arXiv preprint arXiv:2404.16698*, 2024. URL https://arxiv.org/pdf/2404.16698.

Chen Qian, Yufan Dang, Jiahao Li, Wei Liu, Zihao Xie, Yifei Wang, Weize Chen, Cheng Yang, Xin Cong, Xiaoyin Che, Zhiyuan Liu, and Maosong Sun. Experiential Co-Learning of Software-Developing Agents. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024a. URL https://aclanthology.org/2024.acl-long.305/.

Chen Qian, Jiahao Li, Yufan Dang, Wei Liu, YiFei Wang, Zihao Xie, Weize Chen, Cheng Yang, Yingli Zhang, Zhiyuan Liu, and Maosong Sun. Iterative Experience Refinement of Software-Developing Agents. In *arXiv preprint arXiv:2405.04219*, 2024b. URL https://arxiv.org/pdf/2405.04219.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative Agents for Software Development. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024c. URL https://aclanthology.org/2024.acl-long.810/.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-World APIs. In *International Conference on Learning Representations (ICLR)*, 2024a. URL https://iclr.cc/virtual/2024/poster/18267.

Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, Xuanhui Wang, and Michael Bendersky. Large Language Models are Effective Text Rankers with Pairwise Ranking Prompting. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2024b. URL https://aclanthology.org/2024.findings-naacl.97/.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language Models are Unsupervised Multitask Learners. In *OpenAI Blog*, 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Matthew Renze and Erhan Guven. Self-Reflection in LLM Agents: Effects on Problem-Solving Performance. In *arXiv preprint arXiv:2405.06682*, 2024. URL https://arxiv.org/abs/2405.06682.

Toran Bruce Richards. AutoGPT. In *https://github.com/Significant-Gravitas/AutoGPT*, 2023. URL https://github.com/Significant-Gravitas/AutoGPT.

Yangjun Ruan, Chris J. Maddison, and Tatsunori Hashimoto. Observational Scaling Laws and the Predictability of Language Model Performance. In *arXiv preprint arXiv:2405.10938*, 2024. URL https://arxiv.org/pdf/2405.10938.

Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. Are Emergent Abilities of Large Language Models a Mirage? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://papers.neurips.cc/paper_files/paper/2023/file/adc98a266f45005c403b8311ca7e8bd7-Paper-Conference.pdf.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. ToolFormer: Language Models Can Teach Themselves to Use Tools. In *arXiv preprint arXiv:2302.04761*, 2023. URL https://arxiv.org/pdf/2302.04761.

Pier Giuseppe Sessa, Robert Dadashi, Léonard Hussenot, Johan Ferret, Nino Vieillard, Alexandre Ramé, Bobak Shariari, Sarah Perrin, Abe Friesen, Geoffrey Cideron, Sertan Girgin, Piotr Stanczyk, Andrea Michi, Danila Sinopalnikov, Sabela Ramos, Amélie Héliou, Aliaksei Severyn, Matt Hoffman, Nikola Momchev, and Olivier Bachem. BOND: Aligning LLMs with Best-of-N Distillation. In *arXiv preprint arXiv:2407.14622*, 2024. URL https://arxiv.org/abs/2407.14622.

Murray Shanahan, Kyle McDonell, and Laria Reynolds. Role Play with Large Language Models. In *Nature*, 2023. URL https://www.nature.com/articles/s41586-023-06647-8.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/77c33e6a367922d003ff102ffb92b658-Paper-Conference.pdf.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2303.11366.

Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, Elton Zhang, Rewon Child, Reza Yazdani Aminabadi, Julie Bernauer, Xia Song, Mohammad Shoeybi, Yuxiong He, Michael Houston, Saurabh Tiwary, and Bryan Catanzaro. Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model. In *arXiv preprint arXiv:2201.11990*, 2022. URL https://arxiv.org/pdf/2201.11990.

Steven H. Strogatz. Exploring Complex Networks. In *Nature*, 2001. URL https://www.nature.com/inproceedingss/35065725.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive Architectures for Language Agents. In *arXiv preprint arXiv:2309.02427*, 2023. URL https://arxiv.org/pdf/2309.02427.

Yashar Talebirad and Amirhossein Nadiri. Multi-Agent Collaboration: Harnessing the Power of Intelligent LLM Agents. In *arXiv preprint arXiv:2306.03314*, 2023. URL https://arxiv.org/abs/2306.03314.

Xiangru Tang, Anni Zou, Zhuosheng Zhang, Ziming Li, Yilun Zhao, Xingyao Zhang, Arman Cohan, and Mark Gerstein. MedAgents: Large Language Models as Collaborators for Zero-shot Medical Reasoning. In *arXiv preprint arXiv:2311.10537*, 2023. URL https://arxiv.org/pdf/2311.10537.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and Efficient Foundation Language Models. In *arXiv preprint arXiv:2302.13971*, 2023. URL https://arxiv.org/pdf/2302.13971.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is All You Need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, and et al. Grandmaster Level in StarCraft II using Multi-agent Reinforcement Learning. In *Nature*, 2019. URL https://doi.org/10.1038/s41586-019-1724-z.

Haotian Wang, Xiyuan Du, Weijiang Yu, Qianglong Chen, Kun Zhu, Zheng Chu, Lian Yan, and Yi Guan. Learning to Break: Knowledge-Enhanced Reasoning in Multi-Agent Debate System. In *arXiv preprint arXiv:2312.04854*, 2024a. URL https://arxiv.org/pdf/2312.04854.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023a. URL https://aclanthology.org/2023.acl-long.147.pdf.

Lei Wang, Jingsen Zhang, Xu Chen, Yankai Lin, Ruihua Song, Wayne Xin Zhao, and Ji-Rong Wen. RecAgent: A Novel Simulation Paradigm for Recommender Systems. In *arXiv preprint arXiv:2306.02552*, 2023b. URL https://arxiv.org/pdf/2306.02552.

Qineng Wang, Zihao Wang, Ying Su, Hanghang Tong, and Yangqiu Song. Rethinking the Bounds of LLM Reasoning: Are Multi-Agent Discussions the Key? In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024b. URL https://aclanthology.org/2024.acl-long.331/.

Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. Unleashing the Emergent Cognitive Synergy in Large Language Models: A Task-Solving Agent through Multi-Persona Self-Collaboration. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2024c. URL https://aclanthology.org/2024.naacl-long.15/.

Duncan J. Watts and Steven H. Strogatz. Collective Dynamics of Small-World Networks. In *Nature*, 1998. URL https://www.nature.com/inproceedingss/30918#citeas.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent Abilities of Large Language Models. In *Transactions on Machine Learning Research*, 2022a. URL https://arxiv.org/abs/2206.07682.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022b. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

Anita Williams Woolley, Christopher F Chabris, Alex Pentland, Nada Hashmi, and Thomas W Malone. Evidence for a Collective Intelligence Factor in the Performance of Human Groups. In *Science*, 2010. URL https://www.science.org/doi/10.1126/science.1193147.

Peng Xu, Wei Ping, Xianchao Wu, Lawrence McAfee, Chen Zhu, Zihan Liu, Sandeep Subramanian, Evelina Bakhturina, Mohammad Shoeybi, and Bryan Catanzaro. Retrieval Meets Long Context Large Language Models. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2310.03025.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large Language Models as Optimizers. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2309.03409.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/271db9922b8d1f4dd7aaef84ed5ac703-Paper-Conference.pdf.

Zhangyue Yin, Qiushi Sun, Cheng Chang, Qipeng Guo, Junqi Dai, Xuanjing Huang, and Xipeng Qiu. Exchange-of-Thought: Enhancing Large Language Model Capabilities through Cross-Model Communication. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2023. URL https://aclanthology.org/2023.emnlp-main.936/.

Jifan Yu, Zheyuan Zhang, Daniel Zhang-li, Shangqing Tu, Zhanxin Hao, Rui Miao Li, Haoxuan Li, Yuanchun Wang, Hanming Li, Linlu Gong, Jie Cao, Jiayin Lin, Jinchang Zhou, Fei Qin, Haohua Wang, Jianxiao Jiang, Lijun Deng, Yisi Zhan, Chaojun Xiao, Xusheng Dai, Xuan Yan, Nianyi Lin, Nan Zhang, Ruixin Ni, Yang Dang, Lei Hou, Yu Zhang, Xu Han, Manli Li, Juanzi Li, Zhiyuan Liu, Huiqin Liu, and Maosong Sun. From MOOC to MAIC: Reshaping Online Teaching and Learning through LLM-driven Agents. In *arXiv preprint arXiv:2409.03512*, 2024. URL https://arxiv.org/pdf/2409.03512.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D. Goodman. Quiet-STaR: Language Models Can Teach Themselves to Think Before Speaking. In *arXiv preprint arXiv:2403.09629*, 2024. URL https://arxiv.org/abs/2403.09629.

Zheni Zeng, Bangchen Yin, Shipeng Wang, Jiarui Liu, Cheng Yang, Haishen Yao, Xingzhi Sun, Maosong Sun, Guotong Xie, and Zhiyuan Liu. ChatMol: Interactive Molecular Discovery with Natural Language. In *Bioinformatics*, 2024. URL https://doi.org/10.1093/bioinformatics/btae534.

An Zhang, Leheng Sheng, Yuxin Chen, Hao Li, Yang Deng, Xiang Wang, and Tat-Seng Chua. On Generative Agents in Recommendation. In *arXiv preprint arXiv:2310.10108*, 2023. URL https://arxiv.org/pdf/2310.10108.

Bin Zhang, Hangyu Mao, Jingqing Ruan, Ying Wen, Yang Li, Shao Zhang, Zhiwei Xu, Dapeng Li, Ziyue Li, Rui Zhao, Lijuan Li, and Guoliang Fan. Controlling Large Language Model-based Agents for Large-Scale Decision-Making: An Actor-Critic Approach. In *arXiv preprint arXiv:2311.13884*, 2024a. URL https://arxiv.org/pdf/2311.13884.

Yifan Zhang, Yang Yuan, and Andrew Chi-Chih Yao. On the Diagram of Thought. In *arXiv preprint arXiv:2409.10038*, 2024b. URL https://arxiv.org/pdf/2409.10038.

Zheyuan Zhang, Daniel Zhang-Li, Jifan Yu, Linlu Gong, Jinchang Zhou, Zhiyuan Liu, Lei Hou, and Juanzi Li. Simulating Classroom Education with LLM-Empowered Agents. In *arXiv preprint arXiv:2406.19226*, 2024c. URL https://arxiv.org/pdf/2406.19226.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: LLM Agents are Experiential Learners. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2024. URL https://doi.org/10.1609/aaai.v38i17.29936.

Zirui Zhao, Wee Sun Lee, and David Hsu. Large Language Models as Commonsense Knowledge for Large-Scale Task Planning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/65a39213d7d0e1eb5d192aa77e77eeb7-Paper-Conference.pdf.

Wangchunshu Zhou, Yixin Ou, Shengwei Ding, Long Li, Jialong Wu, Tiannan Wang, Jiamin Chen, Shuai Wang, Xiaohua Xu, Ningyu Zhang, Huajun Chen, and Yuchen Eleanor Jiang. Symbolic Learning Enables Self-Evolving Agents. In *arXiv preprint arXiv:2406.18532*, 2024. URL https://arxiv.org/abs/2406.18532.

Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A. Rossi, Somdeb Sarkhel, and Chao Zhang. ToolChain*: Efficient Action Space Navigation in Large Language Models with A* Search. In *arXiv preprint arXiv:2310.13227*, 2024. URL https://arxiv.org/pdf/2310.13227.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jurgen Schmidhuber. Language Agents as Optimizable Graphs. In *International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/pdf/2402.16823.

The appendix of the paper *Scaling Large Language Model-based Multi-Agent Collaboration* presents supplementary materials such as theoretical derivations, dataset descriptions, additional results, and case studies. These comprehensive details are intended for the review phase. The final version of the appendix will be appropriately condensed based on the significance of each section and feedback from the reviewers.

A Theoretical Derivations: Token Complexity Analysis

This section analyzes token consumption complexity in a network, focusing on a mesh structure. A mesh network, with its high interaction density, connects each node to many others, facilitating extensive communication. This makes it ideal for examining the upper bounds of token consumption complexity, as structures with fewer connections will have equal or lower complexities.

We start by calculating token consumption for a single agent in the network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents nodes and $\mathcal{E}$ represents edges. The network scale, n, is the number of nodes ($|\mathcal{V}|$). Other parameters include:

Symbol	Description
t	Task length
p	Profile length
i	Average instruction length
s	Average solution length
m	Maximum interaction rounds between adjacent agents

Without memory control mechanisms, the token consumption for the source executor (agent at the source node) is calculated as:

$$\mathcal{O}(v_1)_{\text{w/o}} = \mathcal{O}(v_1)^{\text{input}}_{\text{w/o}} + \mathcal{O}(v_1)^{\text{output}}_{\text{w/o}} = (t + p) + s \tag{8}$$

This equation represents the source executor's basic needs: understanding the task, knowing its profile (role and tools), and generating a solution, similar to the direct inference process of most LLMs.

Once the source executor generates information, it interacts with an instructor through a connected edge, before the instructor interacts with another executor, involving multiple rounds of reflected instructions and refined solutions. Therefore, for the second agent, token consumption is:

$$\begin{aligned} \mathcal{O}(v_2)_{\text{w/o}} &= (t + p + s) + (mi + (m-1)s) + (ms + (m-1)i) \\ &= t + p + s + (2m-1)(i+s) \end{aligned} \tag{9}$$

This shows that each additional edge in the network increases token consumption by $(2m - 1)(i + s)$.

For the sink agent (the final agent in $\mathcal{G}$), without aggregation mechanisms, token consumption is:

$$\begin{aligned} \mathcal{O}(v_n)^{\text{w/o-agg}}_{\text{w/o}} &= t + p + s + (2m-1)(i+s)|\mathcal{E}| \\ &= t + p + s + (2m-1)(i+s)\frac{n(n-1)}{2} \end{aligned} \tag{10}$$

where $|\mathcal{E}|$ is the number of edges, calculated as $\frac{n(n-1)}{2}$ for a fully connected mesh network.

The sink node aggregates solutions from $n - 1$ previous nodes. Let d be the number of branches aggregated at each step in a hierarchical process. Total token consumption for aggregation is:

$$\mathcal{O}(v_n)^{\text{w/-agg}}_{\text{w/o}} = (2m-1)(i+s)\mathcal{T}(|\bullet v_n|) \tag{11}$$

where $\bullet v$ represents predecessor nodes of v, $\mathcal{T}(n)$ is the number of edges in a d-way tree with n lead nodes:

$$\begin{aligned} \mathcal{T}(|\bullet v_n|) = \mathcal{T}(n-1) &= n - 1 + \frac{n-1}{d} + \frac{n-1}{d^2} + \cdots \\ &= (n-1)\left(1 + \frac{1}{d} + \frac{1}{d^2} + \cdots\right) \\ &= (n-1)\left(\frac{1 - (\frac{1}{d})^{\lceil \log_d (n-1) \rceil}}{1 - \frac{1}{d}}\right) \\ &= \frac{d(n-2)}{d-1} \end{aligned} \tag{12}$$

This formula accounts for cumulative token consumption as solutions are aggregated through the network, considering the branching factor d.

In binary aggregation, where each step combines two branches ($d = 2$), the total token consumption for the sink agent is:

$$\begin{aligned}\mathcal{O}(v_n)_{\text{w/o}} &= \mathcal{O}(v_n)_{\text{w/o}}^{\text{w/o-agg}} + \mathcal{O}(v_n)_{\text{w/o}}^{\text{w/-agg}} \\ &= t + p + s + (2m-1)(i+s)\left(\frac{n(n-1)}{2} + 2(n-2)\right)\end{aligned} \tag{13}$$

Here, $\frac{n(n-1)}{2}$ represents token consumption from interactions across all edges in a fully connected mesh network. The term $2(n-2)$ accounts for binary aggregation, where each step halves the number of nodes at each hierarchy level. This formula illustrates the balance between interaction and aggregation costs: interaction costs grow at a quadratic rate with node count due to the mesh structure, while aggregation costs grow linearly, showing the efficiency of binary aggregation.

Similarly, utilizing the proposed memory control mechanism, the total token consumption for the source agent under minimal context pressure is:

$$\mathcal{O}(v_1)_{\text{w/}} = t + p + s \tag{14}$$

For the second executor, the total token consumption is:

$$\begin{aligned}\mathcal{O}(v_2)_{\text{w/}} &= (t + p + s) + i + (ms + (m-1)i) \\ &= t + p + s + m(i+s)\end{aligned} \tag{15}$$

Each additional edge increases token consumption by $m(i+s)$. Therefore, the sink agent's token consumption, excluding aggregation, is:

$$\begin{aligned}\mathcal{O}(v_n)_{\text{w/}}^{\text{w/o-agg}} &= t + p + s + m(i+s)|\bullet v_2| \\ &= t + p + s + m(i+s)(n-1)\end{aligned} \tag{16}$$

The sink node aggregates $n-1$ solutions, with d branches at each hierarchical step. The total token consumption for aggregation is:

$$\begin{aligned}\mathcal{O}(v_n)_{\text{w/}}^{\text{w/-agg}} &= m(i+s)\mathcal{T}(n-1) \\ &= m(i+s)\frac{d(n-2)}{d-1}\end{aligned} \tag{17}$$

For the binary aggregation setting:

$$\begin{aligned}\mathcal{O}(v_n)_{\text{w/}} &= \mathcal{O}(v_n)_{\text{w/}}^{\text{w/o-agg}} + \mathcal{O}(v_n)_{\text{w/}}^{\text{w/-agg}} \\ &= t + p + s + m(i+s)\left((n-1) + 2(n-2)\right)\end{aligned} \tag{18}$$

In conclusion, for large n, the expressions simplify to:

$$\begin{aligned}\mathcal{O}(v_n)_{\text{w/o}} &\overset{n\gg 1}{\approx} \frac{(2m-1)(i+s)}{2}n^2 \quad && \propto n^2 \\ \mathcal{O}(v_n)_{\text{w/}} &\overset{n\gg 1}{\approx} 3m(i+s)n && \propto n\end{aligned} \tag{19}$$

These indicate quadratic growth without memory control and linear growth with memory control, highlighting its efficiency as n increases.

Going deeper, without the implementation of the proposed mechanism, the total computational complexity involved in token consumption across the network can be expressed as follows:

$$\begin{aligned}\mathcal{O}(\mathcal{V})_{\text{w/o}} &= \mathcal{O}(v_1)_{\text{w/o}} + \mathcal{O}(v_2)_{\text{w/o}} + \cdots + \mathcal{O}(v_n)_{\text{w/o}} \\ &= \frac{(2m-1)(i+s)}{2}\left(1^2 + 2^2 + \cdots + n^2\right) \\ &= \frac{(2m-1)(i+s)}{2}\frac{n(n+1)(2n+1)}{6} \\ &\overset{n\gg 1}{\approx} \frac{(2m-1)(i+s)}{6}n^3 \\ &\propto n^3\end{aligned} \tag{20}$$

From this expression, it is evident that the absence of the mechanism results in a cubic growth rate of token consumption relative to the size of the network n. This cubic complexity signifies substantial computational overhead, limiting the scalability of the network for larger datasets or more extensive applications.

Conversely, when the mechanism is applied, the inference token consumption undergoes a significant transformation:

$$\begin{aligned} \mathcal{O}(\mathcal{V})_{\text{w/}} &= \mathcal{O}(v_1)_{\text{w/}} + \mathcal{O}(v_2)_{\text{w/}} + \cdots + \mathcal{O}(v_n)_{\text{w/}} \\ &= 3m(i+s)\left(1+2+\cdots+n\right) \\ &= 3m(i+s)\frac{n(n+1)}{2} \\ &\overset{n\gg 1}{\approx} \frac{3m(i+s)}{2}n^2 \\ &\propto n^2 \end{aligned} \tag{21}$$

The introduction of the mechanism reduces the computational complexity from cubic to quadratic with respect to n. This notable reduction facilitates enhanced scalability and performance, making it more feasible to implement the network for larger-scale inference tasks. Therefore, this highlights the potential of the mechanism to significantly reduce token consumption during the inference process, thereby paving the way for more efficient and scalable network architectures.

B Supplementary Descriptions: Datasets

MMLU The MMLU dataset is a massive multitask test consisting of multiple-choice questions from various branches of knowledge. The test covers 57 tasks including elementary mathematics, US history, computer science, law, and more. It ranges in difficulty from an elementary level to an advanced professional level, and it tests both world knowledge and problem-solving ability. All 57 tasks and their detailed topics are shown in Figure 9. The initial format of questions is shown in Figure 10.

HumanEval The HumanEval dataset comprises 164 hand-written programming problems, each including a function signature, a docstring, a function body, and multiple unit tests. Problems are designed to test the model's ability to generate functionally correct code from natural language specifications. For instance, the tasks often involve implementing algorithms for sorting, searching, and manipulating data structures such as arrays and strings. An example of the initial prompt of the HumanEval test is shown in Figure 11. Each problem also includes multiple test cases that validate the correctness of the generated code.

SRDD The SRDD dataset is a comprehensive database containing 1,200 software descriptions for automatic software generation. The dataset structure is shown in Figure 12. The construction of this database adhered to the following three-stage strategy for constructing a diverse and unique dataset: 1) Random Sampling: First, ChatGPT is independently inquired multiple times to obtain software information under a certain category, and then the duplication is removed at the token granularity of the software name. 2) Sequential Sampling: Then we add the generated software information in sequence in the form of negative prompts, requiring ChatGPT to continue generating unique software information. 3) Check: Although ChatGPT has been required to follow certain rules when generating, LLM is more likely to be overconfident when generating according to rules than when judging based on rules. Therefore, our last step is to let ChatGPT determine whether the generated software follows the rules. This strategy initially establishes datasets by random sampling some software data, then records existing data, granting ChatGPT autonomy to produce novel entries. SRDD is created with human-designed rules that make the created software easy for researchers to evaluate, for example, the collected software does not need internet or multi-player participation. The length distribution of software descriptions in SRDD is shown in Figure 13. We sought to analyze the effects and semantic features of the generated software descriptions by using t-SNE to perform dimensionality reduction and visualization on the description embedding generated by the OpenAI Ada Model. As demonstrated in figure 14, significant clustering of tasks bearing the same color is observed. It can be concluded that 1) software descriptions of the same category are distributed in clusters, indicating that the generated descriptions are highly related to their categories. 2) Descriptions in different subcategories under the same category are clustered together, such as the game subcategories in the lower right corner. 3) Some subcategories of different categories also show overlaps in the figure, such as Tools&Utilities and Graphics, Schedule and Business, Sports and Sports Game. Such an overlap is comprehensible given the multi-functionality of some software applications that may not be confined to a single classification.

CommonGen-Hard The CommonGen dataset is a constrained text generation task designed to evaluate the ability of generative models in commonsense reasoning. The dataset is composed of 35,141 unique concept sets and corresponding human-annotated sentences that describe everyday scenarios using those concepts. The CommonGen-Hard dataset is a more challenging variant of the original dataset CommonGen. CommonGen-Hard requires models to generate coherent and grammatically correct sentences incorporating 20-30 concepts, as opposed to the original task which presents a set of 3-5 related concepts. This significant increase in the number of concepts tests the model's ability to perform advanced commonsense reasoning, contextual understanding, and creative problem-solving, as it must generate meaningful sentences that encompass a broader range of ideas. Two key challenges of the tests are *rational reasoning* with underlying commonsense knowledge about given concepts, and *compositional generalization* for unseen combination of concepts. Samples shown in Figure 15 include a concept set and the coherent sentences generated.

Licence The four datasets used in this experiment are all licensed under the CC-BY-NC-4.0 license, allowing free use for scientific research.

C Supplementary Experiments: Time Consumption Analysis

To investigate the time costs of MacNet and the underlying mechanisms, we analyzed the results on the SRDD dataset. To maximize the difference in topological properties (*e.g.,* graph density, maximum depth, etc,) the number of nodes is chosen as 50. As mentioned in the mainbody, a topology $\mathcal{G}$ requires at least $2 \times |\mathcal{E}|$ interaction rounds. Therefore, interaction rounds for different types of topologies can be calculated as in Figure 16. After carefully examining the experiment logs, it can be concluded that consumed time is positively correlated with the quantity of interaction rounds. We recorded the average time consumed on each type of topology, as shown in Figure 17.

Similar results can also be obtained from other datasets and topologies. Moreover, we noticed that cost increases exponentially rather than linearly as the number of interaction rounds increases. Consequently, it is suggested that future implementation should carefully balance the cost and performance.

D Supplementary Experiments: Ablation Study

To study the role of profiles in the agent reasoning process within our system, we orchestrated a series of experiments in which the profiles of all agents were left blank. As illustrated in Figure 18, the performance of MacNet deteriorates for an average of 3.75% with the absence of the profiles. This phenomenon suggests that the profile deployment mechanism of MacNet is effective.

Additionally, we conducted experiments utilizing Claude[10] as the base model. The number of nodes was set to 4 and datasets were selected as SRDD and CommonGen, mainly considering costs. Profile deployment and topologies align with the configurations delineated in implementation details. Figure 19 demonstrates that Claude outperforms ChatGPT in these experiments.

E Supplementary Experiments: Case Study

This section presents a case study on software developed, detailing each stage of its lifecycle. The representative software is "*Business Sales Performance Tracker*" with a user's requirement: "*Business Sales Performance Tracker is a software application that helps businesses track and analyze their sales performance. It provides features for inputting sales data, generating reports, and visualizing*

[10]Claude 3 sonnet (until 20240229), by Anthropic.

sales performance metrics. The application also allows businesses to set sales goals and compare actual performance against targets".

Figure 20 illustrates the Business Sales Performance Tracker's user interface. On the top left, a data entry interface is displayed, where users can input sales-related information. This interface allows for the repeated entry of customer names, product names, and sales figures into designated fields. Users can then click the "Add Sales Data" button to integrate this information into the tracking system. For generating comprehensive reports, the user can click the "Generate Report" button. This action produces a statistical report within a terminal window, displaying key metrics such as total revenue, sales growth, conversion rate, average order value, customer acquisition cost, and customer lifetime value. Additionally, a visual report in the form of a histogram is displayed on the right side of the window. The software includes tools in the toolbar, which enable users to customize the histogram's layout and style. These tools also provide options to save and export the graphical data representations.

Figures 21, 22, 23, 24, 25, 26, 27, 28, 29, and 30 provide a comprehensive view of the multi-agent interaction. Each figure captures the detailed dialogue and interactions, showcasing the collaborative efforts and methodologies employed in the development of the software.

Figure 31 illustrates a case of a single-agent generating code on the SRDD dataset. Figures 32, 33, and 34 compare the code generated by our multi-agent system ($|\mathcal{V}|$=50) using the same prompt. It demonstrates that multi-agent collaboration results in multidimensional features (such as multi-file output, code comments, user interface, and operational correctness) accompanied by a significant increase in solution length.

To view additional examples of software developed by MACNET-CHAIN, please refer to Figure 35 for screenshots.

F DISCUSSION: LIMITATIONS AND FUTURE WORK

While our study has thoroughly explored the capabilities of collaborative autonomous agents across various tasks, it is crucial for both researchers and practitioners to remain cognizant of the limitations and risks associated with this study.

Compared to single-agent methods, the iterative interactions between multiple agents inherently demand more tokens and time, leading to increased computational requirements for the backbone models and potential environmental impacts. For example, our extensive experiment spanned more than six weeks and incurred of at least \$3,024.62. While the findings were informative and intriguing, the high resource expenditure raises concerns about the sustainability of future research endeavors. To address this, future research could focus on developing methods that enable agents to achieve equivalent or superior capabilities with fewer interactions. A promising strategy is to avoid full-graph inference by utilizing only a subset of the graph, such as identifying the best sub-team to execute the task.

We examined six representative topologies and identified a promising architectural direction through observed phenomena. However, within the vast space of network structures, identifying the theoretically optimal collaborative network of agents without bias remains a challenge. Further exploration into this optimal collaborative network is an interesting direction for future research. Moreover, there is significant value in exploring collaborative mechanisms, such as dynamically generating and assigning agents (including personalized profiles, external tools, multi-step planning, foundation models, and finer-grained labor division), and enhancing inference coordination (*e.g.,* efficient routing strategies, information transmission mechanisms, and long-context management).

In agents' reasoning, the aggregation of multiple solutions at graph nodes presents a complex challenge. The current strategy of combining strengths and eliminating weaknesses offers foundational insights but may fall short due to model hallucinations, potentially leading to performance degradation. We recommend designing the topology to minimize convergent nodes, while also developing a more robust aggregation strategy to effectively address this issue.

The performance of multi-agent collaboration, given its additional factors, is inherently more unpredictable than traditional scaling. We minimize bias through general designs and repeated experiments, but future work should consider more mature patterns and higher-quality metrics. As current technology lacks precise automated evaluation systems for complex tasks (*e.g.,* software development and creative writing), manual verification becomes labor-intensive and impractical for large-scale datasets. This study focuses on objective and critical dimensions, such as comprehensive software indicators considering completeness, executability, and consistency. Future research should investigate finer-grained dimensions to enhance the objectivity and quantifiability of performance evaluations, including solutions' functionalities, robustness, safety, and user-friendliness.

Given the nascent stage of multi-agent collaboration models, most relevant studies focus on inference. When faced with diverse tasks, current methods handle each task independently due to the lack of methodologies that effectively incorporate past experiences. This inexperience often results in repetitive errors or unnecessary trial-and-error processes in multi-step tasks, requiring additional human intervention, especially in real-world applications. Therefore, multi-agent collaborative learning is an urgent area for research, promising more efficient cross-task inference and reduced resource consumption.

However, we believe that these potential limitations serve as inspiration for future research directions and can be effectively mitigated by engaging a broader, technically proficient audience. We expect that our findings will provide valuable insights into enhancing collaborative learning and reasoning in the ever-evolving dynamics of LLM-powered agents.

G Reproducibility: Software and Data

The `SupplementaryMaterials.zip` file contains detailed configuration guidelines, execution commands, source code, and datasets used in this study, along with additional resources. These materials are meticulously curated to enable the replication of all data presented in our paper. They have been rigorously validated, with successful installation and testing conducted by multiple testers, ensuring compatibility with both Windows and Mac OS systems. This comprehensive preparation significantly enhances the reproducibility of our findings. All materials will be publicly accessible on GitHub to support future research endeavors.

H AI Assistants

ChatGPT[11] was used purely with the language of the paper during the writing process, including spell-checking and paraphrasing the authors' original content, without suggesting new content. Any content generated with the assistant underwent meticulous manual review and subsequently received final approval from the authors.

[11]https://chat.openai.com/

Task	Tested Concepts	Supercategory
Abstract Algebra	Groups, rings, fields, vector spaces, ...	STEM
Anatomy	Central nervous system, circulatory system, ...	STEM
Astronomy	Solar system, galaxies, asteroids, ...	STEM
Business Ethics	Corporate responsibility, stakeholders, regulation, ...	Other
Clinical Knowledge	Spot diagnosis, joints, abdominal examination, ...	Other
College Biology	Cellular structure, molecular biology, ecology, ...	STEM
College Chemistry	Analytical, organic, inorganic, physical, ...	STEM
College Computer Science	Algorithms, systems, graphs, recursion, ...	STEM
College Mathematics	Differential equations, real analysis, combinatorics, ...	STEM
College Medicine	Introductory biochemistry, sociology, reasoning, ...	Other
College Physics	Electromagnetism, thermodynamics, special relativity, ...	STEM
Computer Security	Cryptography, malware, side channels, fuzzing, ...	STEM
Conceptual Physics	Newton's laws, rotational motion, gravity, sound, ...	STEM
Econometrics	Volatility, long-run relationships, forecasting, ...	Social Sciences
Electrical Engineering	Circuits, power systems, electrical drives, ...	STEM
Elementary Mathematics	Word problems, multiplication, remainders, rounding, ...	STEM
Formal Logic	Propositions, predicate logic, first-order logic, ...	Humanities
Global Facts	Extreme poverty, literacy rates, life expectancy, ...	Other
High School Biology	Natural selection, heredity, cell cycle, Krebs cycle, ...	STEM
High School Chemistry	Chemical reactions, ions, acids and bases, ...	STEM
High School Computer Science	Arrays, conditionals, iteration, inheritance, ...	STEM
High School European History	Renaissance, reformation, industrialization, ...	Humanities
High School Geography	Population migration, rural land-use, urban processes, ...	Social Sciences
High School Gov't and Politics	Branches of government, civil liberties, political ideologies, ...	Social Sciences
High School Macroeconomics	Economic indicators, national income, international trade, ...	Social Sciences
High School Mathematics	Pre-algebra, algebra, trigonometry, calculus, ...	STEM
High School Microeconomics	Supply and demand, imperfect competition, market failure, ...	Social Sciences
High School Physics	Kinematics, energy, torque, fluid pressure, ...	STEM
High School Psychology	Behavior, personality, emotions, learning, ...	Social Sciences
High School Statistics	Random variables, sampling distributions, chi-square tests, ...	STEM
High School US History	Civil War, the Great Depression, The Great Society, ...	Humanities
High School World History	Ottoman empire, economic imperialism, World War I, ...	Humanities
Human Aging	Senescence, dementia, longevity, personality changes, ...	Other
Human Sexuality	Pregnancy, sexual differentiation, sexual orientation, ...	Social Sciences
International Law	Human rights, sovereignty, law of the sea, use of force, ...	Humanities
Jurisprudence	Natural law, classical legal positivism, legal realism, ...	Humanities
Logical Fallacies	No true Scotsman, base rate fallacy, composition fallacy, ...	Humanities
Machine Learning	SVMs, VC dimension, deep learning architectures, ...	STEM
Management	Organizing, communication, organizational structure, ...	Other
Marketing	Segmentation, pricing, market research, ...	Other
Medical Genetics	Genes and cancer, common chromosome disorders, ...	Other
Miscellaneous	Agriculture, Fermi estimation, pop culture, ...	Other
Moral Disputes	Freedom of speech, addiction, the death penalty, ...	Humanities
Moral Scenarios	Detecting physical violence, stealing, externalities, ...	Humanities
Nutrition	Metabolism, water-soluble vitamins, diabetes, ...	Other
Philosophy	Skepticism, phronesis, skepticism, Singer's Drowning Child, ...	Humanities
Prehistory	Neanderthals, Mesoamerica, extinction, stone tools, ...	Humanities
Professional Accounting	Auditing, reporting, regulation, valuation, ...	Other
Professional Law	Torts, criminal law, contracts, property, evidence, ...	Humanities
Professional Medicine	Diagnosis, pharmacotherapy, disease prevention, ...	Other
Professional Psychology	Diagnosis, biology and behavior, lifespan development, ...	Social Sciences
Public Relations	Media theory, crisis management, intelligence gathering, ...	Social Sciences
Security Studies	Environmental security, terrorism, weapons of mass destruction, ...	Social Sciences
Sociology	Socialization, cities and community, inequality and wealth, ...	Social Sciences
US Foreign Policy	Soft power, Cold War foreign policy, isolationism, ...	Social Sciences
Virology	Epidemiology, coronaviruses, retroviruses, herpesviruses, ...	Other
World Religions	Judaism, Christianity, Islam, Buddhism, Jainism, ...	Humanities

Figure 9: Tasks of the MMLU dataset.

MMLU Prompt

The following are multiple-choice questions (with answers) about abstract algebra.
Find the degree for the given field extension Q(sqrt(2), sqrt(3), sqrt(18)) over Q.
A. 0
B. 4
C. 2
D. 6
Answer:

Figure 10: The official prompt of the MMLU dataset.

HumanEval Prompt

```python
from typing import List


def below_zero(operations: List[int]) -> bool:
    """ You're given a list of deposit and withdrawal operations on a bank
        account that starts with zero balance. Your task is to detect if at any
        point the balance of account falls below zero, and at that point function
         should return True. Otherwise it should return False.
    >>> below_zero([1, 2, 3])
    False
    >>> below_zero([1, 2, -4, 5])
    True
    """
```

Figure 11: The official prompt of the HumanEval dataset.

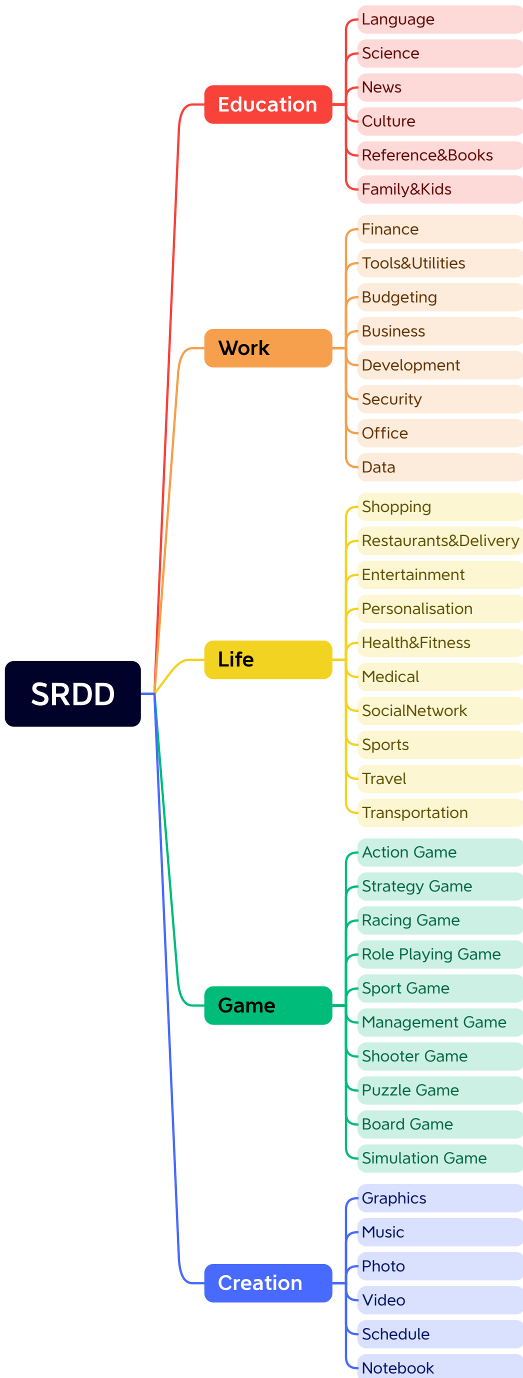


Figure 12: The hierarchy of the SRDD dataset.

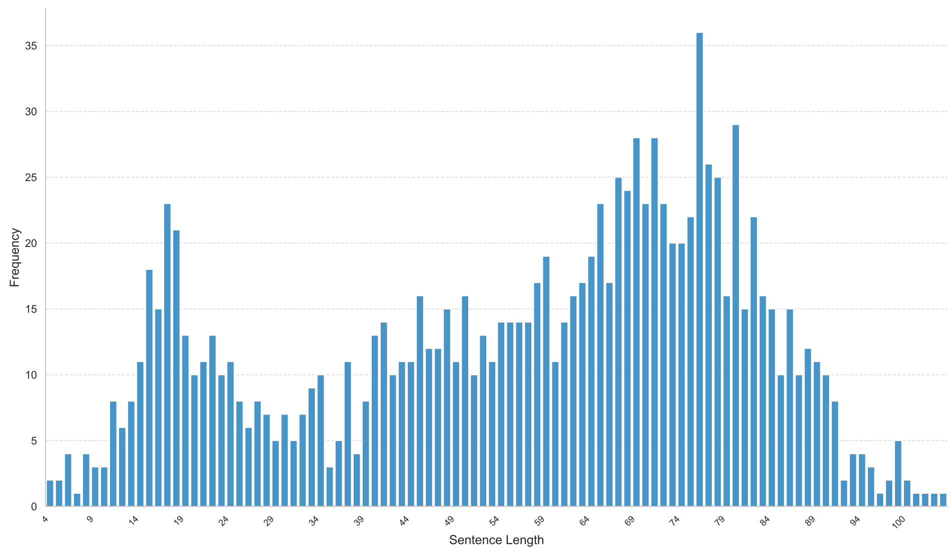


Figure 13: The software description length distribution in SRDD.

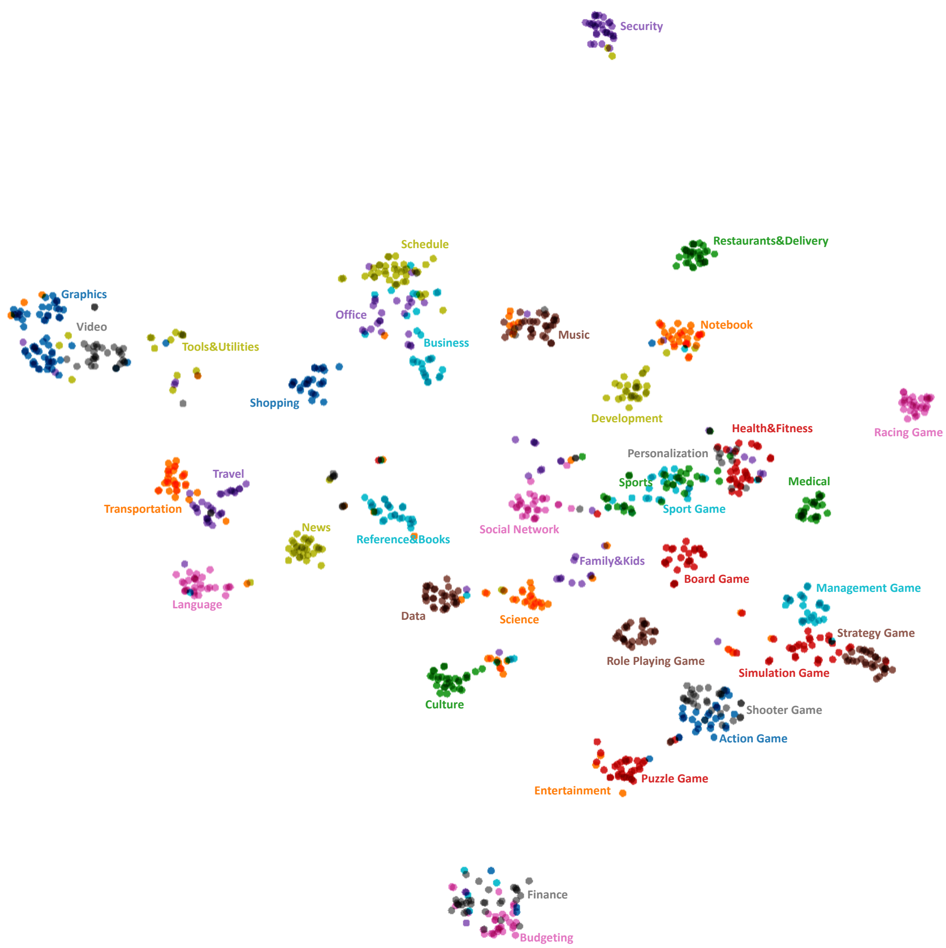


Figure 14: The t-SNE visualization of the SRDD task embeddings via the ada embedding model.

CommonGen Example

Concept set:
["oxidation", "cup", "mower", "space", "window", "arch", "pineapple", "spatula", "smoke", "tie", "gun", "insert", "boat", "use", "tomato", "chest", "vest", "bartender", "move", "microphone", "axis"]
Coherent sentence:
In order to achieve success, one must be persistent in their efforts, just like a mower persistently cuts through the grass. It is important to keep pushing forward, even in the face of adversity, just like the smoke that rises even when the fire is trying to fade. Encouraging others to do the same can lead to a more positive and productive environment. It takes a daring attitude to step out of one's comfort zone and take risks, just like a boat daringly sailing into the vast space of the ocean. By being persistent, encouraging, and daring, one can overcome obstacles and achieve their goals, just like a pineapple growing steadily on its plant, even when it faces harsh weather conditions. I draw inspiration from pushing the boundaries of traditional art forms and exploring new techniques. It is through this bold approach that I am able to create truly unique and inspiring pieces. In my latest piece, I incorporated elements of oxidation to create a stunning visual representation of change and transformation. The image of a cup, delicately balanced on the edge of a window, symbolizes the delicate balance of life. I used a mower to carve intricate patterns into the grass, turning the lawn into a work of art. The vast expanse of space serves as a backdrop for my imagination to run wild, unrestricted by earthly limitations. I adorned the archway with vibrant pineapple sculptures, adding a touch of tropical flair to the serene surroundings. A spatula became my tool of choice as I manipulated smoke to create ethereal and ephemeral forms that danced in the air. As I carefully inserted a boat into the space, signifying a journey towards new horizons and discoveries, with each deliberate move, I guided the piece towards its final form, shaping it with the precision of a sculptor. The gun, once a symbol of destruction, was transformed into a statement of peace and reconciliation. The microphone, positioned at the axis of the composition, captured the subtle nuances and whispers of the artwork, amplifying its impact. In the end, I stood back and admired the culmination of my efforts, a testament to the power of daring creativity and fearless self-expression.

Figure 15: One example of CommonGen's concept-sets and the coherent sentences generated by MACNET.

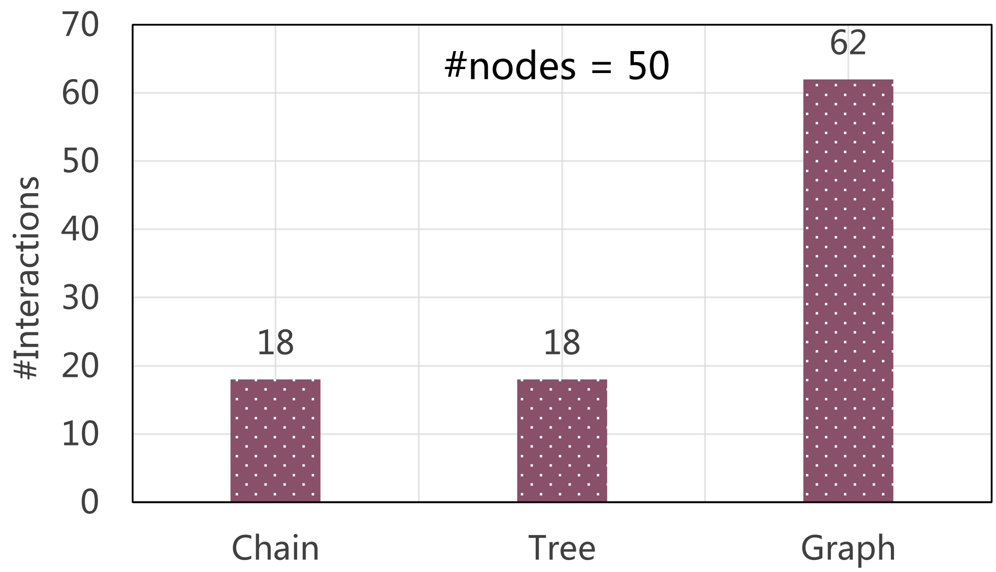


Figure 16: The quantity of interaction rounds in Chain, Tree, and Graph topologies.

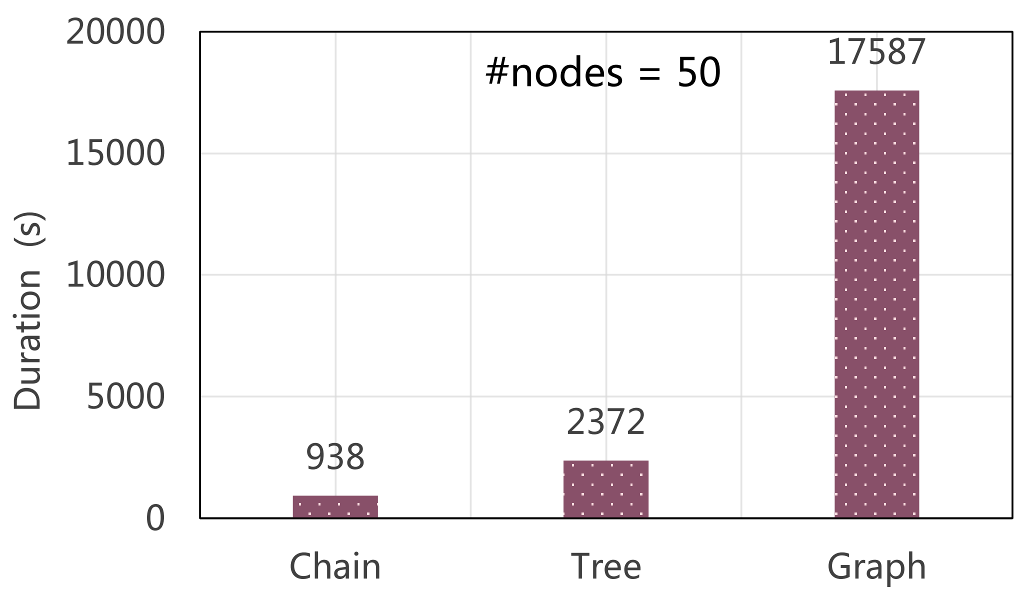


Figure 17: Average time consumed (duration) under different topologies.

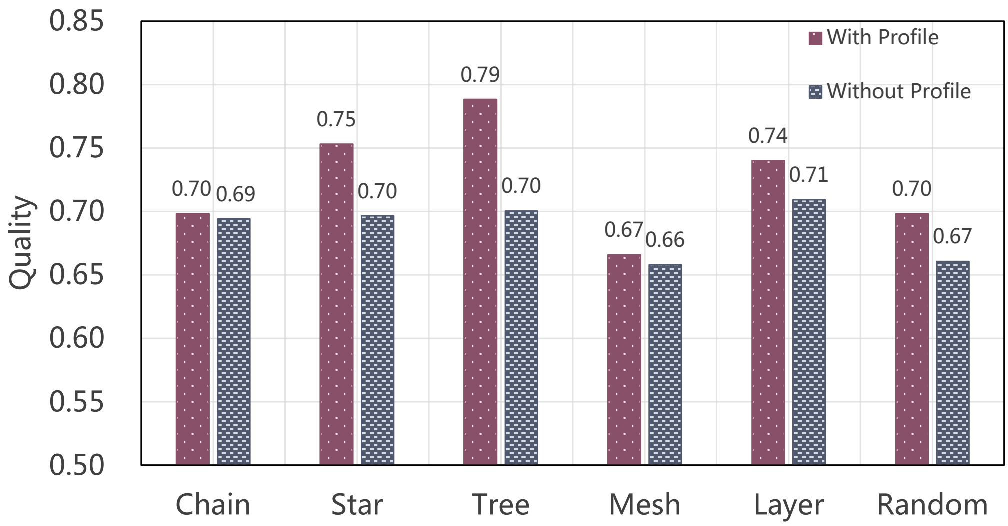


Figure 18: Ablation study on profiles under different topologies.

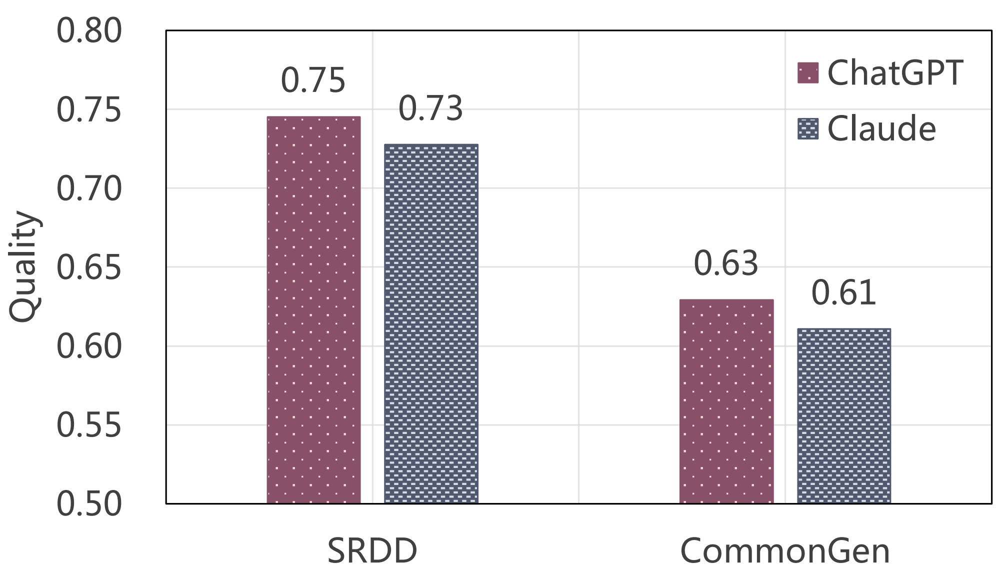


Figure 19: Performances of Claude and ChatGPT on SRDD and CommonGen-Hard datasets.

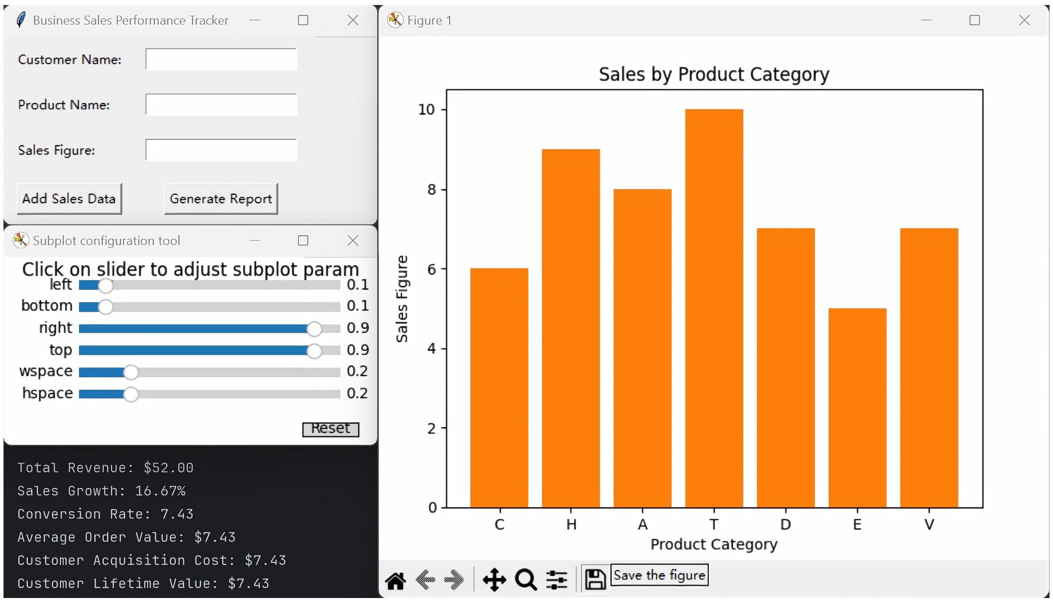


Figure 20: Demonstration of the "Business Sales Performance Tracker" software developed by MACNET-CHAIN.

Profiles

: Instructor : Executor

: You are a highly skilled programmer, ready to embark on the development of a Business application. Your first notable characteristic is your exceptional ability to design effective algorithms and analyze their time and space complexity. With your keen understanding of algorithm design and analysis, you possess the expertise to create efficient and optimized solutions for complex problems. Your meticulous approach ensures that the algorithms you design are not only effective but also scalable, taking into consideration the time and space constraints of the application. In addition to your algorithmic prowess, you possess a second characteristic that sets you apart – software training and education. You have a natural talent for imparting knowledge and providing software training to both users and team members. Your ability to break down complex concepts into easily understandable modules allows you to effectively educate others on the intricacies of the software. Whether it's conducting workshops, creating comprehensive training materials, or offering one-on-one guidance, you excel at empowering individuals to confidently navigate the application. Lastly, your expertise extends to the formulation and execution of software version control strategies. You understand the importance of maintaining a well-organized and controlled development environment. With your meticulous attention to detail, you ensure that the software version control strategy is carefully crafted to facilitate seamless collaboration among team members. You implement best practices, such as branching and merging, to ensure the smooth integration of code changes. Your ability to navigate complex version control systems allows you to effectively manage multiple versions of the software, ensuring stability and reliability throughout the development process. As a programmer poised to develop a Business application, your exceptional skills in algorithm design and analysis, software training and education, and software version control strategy make you an invaluable asset. Your ability to create efficient algorithms, educate others on software usage, and maintain a controlled development environment sets the stage for a successful and impactful application.

: As a programmer poised to develop a Business application, you possess a remarkable proficiency in multiple programming languages. Your mastery of languages like Python, Java, C++, JavaScript, and more, allows you to effortlessly navigate through various coding frameworks and libraries, ensuring a seamless development process. Beyond your programming language expertise, you have also honed your skills in leadership development within software teams. Your ability to cultivate and develop leadership qualities among your team members is commendable. You inspire and motivate your colleagues, fostering a collaborative and efficient work environment. Your guidance and mentorship empower your team to deliver high-quality software solutions, meeting and exceeding client expectations. Furthermore, you excel in providing software user training and support. Understanding the importance of user-friendly applications, you invest time and effort in ensuring that end-users are well-equipped to utilize the software effectively. Your comprehensive training sessions and ongoing support enable users to navigate the application effortlessly, maximizing its potential and enhancing their productivity. In summary, as a programmer poised to develop a Business application, your mastery of multiple programming languages, leadership development skills, and dedication to software user training make you an invaluable asset. Your ability to seamlessly navigate through coding languages, cultivate leadership qualities, and provide exceptional user support ensures the successful development and implementation of Business applications.

Figure 21: Assign profiles to agents.

Agent Interaction in SRDD

: Instructor : Executor

: We have decided to complete the task through a executable software with multiple files implemented via Python. As the Programmer, to satisfy the new user's demands, you should write one or multiple files and make sure that every detail of the architecture is, in the end, implemented as code. The software should be equipped with graphical user interface (GUI) so that user can visually and graphically use it; so you must choose a GUI framework (e.g., in Python, you can implement GUI via tkinter, Pygame, Flexx, PyGUI, etc.).
Think step by step and reason yourself to the right decisions to make sure we get it right.
You will first lay out the names of the core classes, functions, methods that will be necessary, as well as a quick comment on their purpose.

:

main.py

```python
import tkinter as tk
from tkinter import messagebox
from sales_data import SalesData
from sales_report import SaleReport
class SalesPerformanceTracker:
    def __init__(self, root):
        self.root = root
        self.root.title("BusinessSalesPerformanceTracker")
        self.sales_data = SalesData()
        self.create_widgets()
    def create_widgets(self):
        # Create and configure input fields
        self.customer_name_label = tk.Label(self.root, text="CustomerName:"
            )
        self.customer_name_label.grid(row=0, column=0, padx=10, pady=10,
            sticky=tk.W)
        self.customer_name_entry = tk.Entry(self.root)
        self.customer_name_entry.grid(row=0, column=1, padx=10, pady=10)
        self.product_name_label = tk.Label(self.root, text="ProductName:")
        self.product_name_label.grid(row=1, column=0, padx=10, pady=10,
            sticky=tk.W)
        self.product_name_entry = tk.Entry(self.root)
        self.product_name_entry.grid(row=1, column=1, padx=10, pady=10)
        self.sales_figure_label = tk.Label(self.root, text="SalesFigure:")
        self.sales_figure_label.grid(row=2, column=0, padx=10, pady=10,
            sticky=tk.W)
        self.sales_figure_entry = tk.Entry(self.root)
        self.sales_figure_entry.grid(row=2, column=1, padx=10, pady=10)
    def add_sales_data(self):
        customer_name = self.customer_name_entry.get()
        product_name = self.product_name_entry.get()
        sales_figure = self.sales_figure_entry.get()
        if customer_name and product_name and sales_figure:
            self.sales_data.add_data(customer_name, product_name, float(
                sales_figure))
            messagebox.showinfo("Success", "Salesdataaddedsuccessfully!")
        else:
...
```

Figure 22: Agent interaction, part 1.

Agent Interaction in SRDD

: Executor

main.py

```
...

        else:
            messagebox.showerror("Error", "Pleasefillinallfields.")
        self.clear_input_fields()
    def generate_report(self):
        report = SalesReport(self.sales_data)
        report.generate()
    def clear_input_fields(self):
        self.customer_name_entry.delete(0, tk.END)
        self.product_name_entry.delete(0, tk.END)
        self.sales_figure_entry.delete(0, tk.END)
if __name__ == "__main__":
    root = tk.Tk()
    app = SalesPerformanceTracker(root)
    root.mainloop()
```

sales_data.py

```
class SalesData:
    def __init__(self):
        self.data = []
    def add_data(self, customer_name, product_name, sales_figure):
        self.data.append({
            "customer_name": customer_name,
            "product_name": product_name,
            "sales_figure": sales_figure
        })
    def get_total_revenue(self):
        return sum(data["sales_figure"] for data in self.data)
    def get_sales_growth(self):
        if len(self.data) < 2:
            return 0
        first_sales = self.data[0]["sales_figure"]
        last_sales = self.data[-1]["sales_figure"]
        return (last_sales - first_sales) / first_sales * 100
    def get_conversion_rate(self):
        if len(self.data) == 0:
            return 0
        total_sales = sum(data["sales_figure"] for data in self.data)
        total_customers = len(self.data)
        return total_sales / total_customers
    def get_average_order_value(self):
        if len(self.data) == 0:
            return 0
        total_sales = sum(data["sales_figure"] for data in self.data)
        total_orders = len(self.data)
        return total_sales / total_orders
...
```

Figure 23: Agent interaction, part 2.

Agent Interaction in SRDD

: Executor

sales_data.py

```python
...

    def get_customer_acquisition_cost(self):
        if len(self.data) == 0:
            return 0
        total_sales = sum(data["sales_figure"] for data in self.data)
        total_customers = len(self.data)
        return total_sales / total_customers
    def get_customer_lifetime_value(self):
        if len(self.data) == 0:
            return 0
        total_sales = sum(data["sales_figure"] for data in self.data)
        total_customers = len(self.data)
        return total_sales / total_customers
    def get_sales_by_product_category(self):
        sales_by_category = {}
        for data in self.data:
            product_name = data["product_name"]
            sales_figure = data["sales_figure"]
            if product_name in sales_by_category:
                sales_by_category[product_name] += sales_figure
            else:
                sales_by_category[product_name] = sales_figure
        return sales_by_category
```

sales_report.py

```python
import matplotlib.pyplot as plt
class SalesReport:
    def __init__(self, sales_data):
        self.sales_data = sales_data
    def generate(self):
        pass
...
```

Figure 24: Agent interaction, part 3.

Agent Interaction in SRDD

: Executor

sales_data.py

```python
...

    def generate_summary_report(self):
        total_revenue = self.sales_data.get_total_revenue()
        sales_growth = self.sales_data.get_sales_growth()
        conversion_rate = self.sales_data.get_conversion_rate()
        average_order_value = self.sales_data.get_average_order_value()
        customer_acquisition_cost = self.sales_data.
            get_customer_acquisition_cost()
        customer_lifetime_value = self.sales_data.
            get_customer_lifetime_value()
        report = f"SummaryReport:\n\n"
        report += f"TotalRevenue:${total_revenue:.2f}\n"
        report += f"SalesGrowth:{sales_growth:.2f}%\n"
        report += f"ConversionRate:{conversion_rate:.2f}\n"
        report += f"AverageOrderValue:${average_order_value:.2f}\n"
        report += f"CustomerAcquisitionCost:${customer_acquisition_cost:.2f
            }\n"
        report += f"CustomerLifetimeValue:${customer_lifetime_value:.2f}\n"
        print(report)
    def generate_sales_by_product_category_report(self):
        sales_by_category = self.sales_data.get_sales_by_product_category()
        categories = list(sales_by_category.keys())
        sales_figures = list(sales_by_category.values())
        plt.bar(categories, sales_figures)
        plt.xlabel("ProductCategory")
        plt.ylabel("SalesFigure")
        plt.title("SalesbyProduct Category")
        plt.show()
    def generate_sales_by_product_category_report(self):
        sales_by_category = self.sales_data.get_sales_by_product_category()
        categories = list(sales_by_category.keys())
        sales_figures = list(sales_by_category.values())
        plt.bar(categories, sales_figures)
        plt.xlabel("ProductCategory")
        plt.ylabel("SalesFigure")
        plt.title("SalesbyProduct Category")
        plt.show()
```

: The main.py file contains the main application class `SalesPerformanceTracker`, which handles the GUI and user interactions. It imports the `SalesData` class from sales_data.py and the `SalesReport` class from sales_report.py. The `add_sales_data` method is responsible for...

Figure 25: Agent interaction, part 4.

Agent Interaction in SRDD

: Instructor : Executor

:

Unimplemented File

```
sales_report.py
```

In our software, each file must strictly follow a markdown code block format, where the following tokens must be replaced such that "FILENAME" is the lowercase file name including the file extension, "LANGUAGE" in the programming language, "DOCSTRING" is a string literal specified in source code that is used to document a specific segment of code, and "CODE" is the original code:
FILENAME
```LANGUAGE
'''
DOCSTRING
'''
CODE
```
As the Programmer, to satisfy the complete function of our developed software, you have to implement all methods in the `sales_report.py` file which contains an unimplemented class. Now, implement all methods of the `sales_report.py` and all other codes needed, then output the fully implemented codes, strictly following the required format.

:

sales_report.py

```python
...

def __init__(self, sales_data):
    self.sales_data = sales_data
def generate(self):
    self.generate_summary_report()
    self.generate_sales_by_product_category_report()
def generate_summary_report(self):

...
```

Code Diff on sales_report.py

```
 def generate(self):
-    pass
+    self.generate_summary_report()
+    self.generate_sales_by_product_category_report()
```

Figure 26: Agent interaction, part 5.

Agent Interaction in SRDD

: Instructor : Executor

---

:

{Code}

<Comment>

The highest priority comment is that the `get_customer_lifetime_value` method in the `SalesData` class is currently returning the same value as the `get_customer_acquisition_cost` method. This is incorrect as the customer lifetime value should be calculated differently.

To fix this, you need to update the logic in the `get_customer_lifetime_value` method to calculate the average sales per customer over the lifetime of the business. You can do this by dividing the total revenue by the total number of customers.

Here's the revised code for the `get_customer_lifetime_value` method:

sales_data.py

```python
def get_customer_lifetime_value(self):
    if len(self.data) == 0:
        return 0
    total_sales = sum(data["sales_figure"] for data in self.data)
    total_customers = len(set(data["customer_name"] for data in self.data)
        )
    return total_sales / total_customers
```

Please make this change and ensure that the `get_customer_lifetime_value` method returns the correct value.

:

sales_data.py

```python
...

def get_customer_acquisition_cost(self):
    if len(self.data) == 0:
        return 0
    total_sales = sum(data["sales_figure"] for data in self.data)
    total_customers = len(self.data)
    return total_sales / total_customers
def get_customer_lifetime_value(self):
    if len(self.data) == 0:
        return 0
    total_sales = sum(data["sales_figure"] for data in self.data)
    total_customers = len(set(data["customer_name"] for data in self.data)
        )
    return total_sales / total_customers
...
```

Figure 27: Agent interaction, part 6. To save space, we have omitted the specific code with `{Code}`.

Agent Interaction in SRDD

: Instructor : Executor

:

{Code}

The highest priority comment is that the GUI elements of buttons for "Add Sales Data" and "Generate Report" are not implemented in the `SalesPerformanceTracker` class in main.py. To make the software user-friendly and provide intuitive controls for shape manipulation, the GUI elements need to be created.

To fix this, you need to implement the GUI layout and controls in the `SalesPerformanceTracker` class.

For example:

main.py

```
    self.add_button = tk.Button()
    self.add_button.grid()
    self.generate_report_button = tk.Button()
    self.generate_report_button.grid()
```

Make sure to replace the placeholder code with the actual implementation of the GUI elements and their corresponding event bindings and callback functions. Once these changes are made, the software will have a user-friendly interface with intuitive controls.

:

main.py

```
...

    self.sales_figure_entry = tk.Entry(self.root)
    self.sales_figure_entry.grid(row=2, column=1, padx=10, pady=10)
    # Create buttons
    self.add_button = tk.Button(self.root, text="AddSalesData", command=
        self.add_sales_data)
    self.add_button.grid(row=3, column=0, padx=10, pady=10)
    self.generate_report_button = tk.Button(self.root, text="Generate
        Report", command=self.generate_report)
    self.generate_report_button.grid(row=3, column=1, padx=10, pady=10)
def add_sales_data(self):
    customer_name = self.customer_name_entry.get()
    product_name = self.product_name_entry.get()
...
```

Figure 28: Agent interaction, part 7.

Agent Interaction in SRDD

: Instructor : Executor

:

Test Reports of Source Codes:

Test Error on sales_report.py

```
Traceback (most recent call last):
  File "sales_report.py", line 6, in <module>
    import matplotlib.pyplot as plt
ModuleNotFoundError: No module named 'matplotlib'
```

Error Summary of Test Reports: "nothing need to do on codes, install the missing module" Note that each file must strictly follow ...... Now, use the format exemplified above and modify the problematic codes based on the error summary. Output the codes that you fixed based on the test reported and corresponding explanations (strictly follow the format defined above, including FILENAME, LANGUAGE, DOCSTRING and CODE; incomplete "TODO" codes are strictly prohibited). If no bugs are reported, please return only one line like "<INFO> Finished".

:

<Action> Install matplotlib package

```
> pip3 install matplotlib
Collecting matplotlib
.....
Downloading matplotlib-3.8.2-cp39-cp39-macosx_11_0_arm64.whl (7.5 MB)
   --------- 7.5/7.5 MB 7.6 MB/s eta 0:00:00
Installing collected packages: matplotlib
Successfully installed matplotlib-3.8.2
```

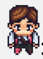
sales_report.py

```
import matplotlib.pyplot as plt
class SalesReport:
    def __init__(self, sales_data):
        self.sales_data = sales_data

...
```

Figure 29: Agent interaction, part 8.

Agent Interaction in SRDD

: Instructor : Executor

---

:

Test Error on main.py

```
Test Reports of Source Codes:
Traceback (most recent call last):
  File "main.py", line 9, in <module>
    from sales_report import SaleReport
ImportError: cannot import name 'SaleReport' from 'sales_report' (
     sales_report.py)
```

Error Summary of Test Reports:
Based on the test report, the bug causing the problem is an import error in the main.py file. The error message states that it cannot import the name `SaleReport` from the `sales_report` module. This suggests that there is an issue with the import statement in the main.py file. To fix this bug, you need to import the correct name from the `sales_report` module.
Note that each file must strictly follow ...... Now, use the format exemplified above and modify the problematic codes based on the error summary. Output the codes that you fixed based on the test reported and corresponding explanations (strictly follow the format defined above, including FILENAME, LANGUAGE, DOCSTRING and CODE; incomplete "TODO" codes are strictly prohibited). If no bugs are reported, please return only one line like "<INFO> Finished".

:

main.py

```
'''
Business Sales Performance Tracker
Author: MACNET-CHAIN
'''
import tkinter as tk
from tkinter import messagebox
from sales_data import SalesData
from sales_report import SalesReport
class SalesPerformanceTracker:
    def __init__(self, root):
        self.root = root
...
```

Code Diff on main.py

```
  from sales_data import SalesData
- from sales_report import SaleReport
+ from sales_report import SalesReport
```

Figure 30: Agent interaction, part 9.

Single-Agent Generated Software

Prompt: A software application that allows users to find local sports matches and events in their area. Users can search for specific sports, view upcoming matches, and connect with other participants.

main.py

```python
class Event:
    def __init__(self, sport, location, date, participants):
        self.sport = sport
        self.location = location
        self.date = date
        self.participants = participants

class SportsApp:
    def __init__(self):
        self.events = []

    def add_event(self, event):
        self.events.append(event)

    def search_events(self, sport):
        matching_events = []
        for event in self.events:
            if event.sport == sport:
                matching_events.append(event)
        return matching_events

    def view_upcoming_events(self):
        upcoming_events = []
        for event in self.events:
            if event.date >= datetime.now().date():
                upcoming_events.append(event)
        return upcoming_events

    def connect_with_participants(self, event):
        pass

app = SportsApp()
event1 = Event("Football", "LocalStadium", datetime(2022, 9, 15), ["John",
     "Mike", "Sarah"])
event2 = Event("Basketball", "CommunityCenter", datetime(2022, 9, 20), ["
    David", "Emily"])
app.add_event(event1)
app.add_event(event2)

football_events = app.search_events("Football")
for event in football_events:
    print(event.sport, event.location, event.date)

upcoming_events = app.view_upcoming_events()
for event in upcoming_events:
    print(event.sport, event.location, event.date)

participants = app.connect_with_participants(event1)
for participant in participants:
    print(participant)
```

Figure 31: Codes generated by single-agent system.

Multi-Agent Generated Software

Prompt: A software application that allows users to find local sports matches and events in their area. Users can search for specific sports, view upcoming matches, and connect with other participants.

database.py

```python
'''
Database module to manage sports events. It includes methods to add events
    , find matches by sport, get upcoming matches by location, and
    retrieve participants by match ID.
'''
from event import Event
class Database:
    def __init__(self):
        self.events = []
    def find_matches(self, sport):
        return [event for event in self.events if event.sport == sport]
    def add_event(self, event):
        self.events.append(event)
    def get_upcoming_matches(self, location):
        return [event for event in self.events if event.location ==
            location]
    def get_participants(self, match_id):
        for event in self.events:
            if event.id == match_id:
                return event.participants
        return None
```

event.py

```python
'''
This file contains the Event class. It initializes with an ID, sport,
    location, date, and time, and maintains a list of participants. The
    add_participant method adds a participant to the event, and the
    __str__ method provides a string representation of the event's
    details.
'''
class Event:
    def __init__(self, id, sport, location, date, time):
        self.id = id
        self.sport = sport
        self.location = location
        self.date = date
        self.time = time
        self.participants = []
    def add_participant(self, participant):
        self.participants.append(participant)
    def __str__(self):
        return f"ID:{self.id},Sport:{self.sport},Location:{self.location},
            Date:{self.date},Time:{self.time}"
```

Figure 32: Codes generated by MACNET, part 1.

Multi-Agent Generated Software

sports_app.py

```
'''
Sports Application module. The `SportsApp` class integrates a `Database`
    and `UserInterface`. It runs a loop offering options to search
    matches, view upcoming matches, or connect with participants,
    handling user input and displaying results accordingly.
'''
from database import Database
from user_interface import UserInterface
class SportsApp:
    def __init__(self):
        self.database = Database()
        self.user_interface = UserInterface()
    def run(self):
        while True:
            self.user_interface.display_menu()
            choice = self.user_interface.get_choice()
            if choice == 1:
                self.search_matches()
            elif choice == 2:
                self.view_upcoming_matches()
            elif choice == 3:
                self.connect_with_participants()
            elif choice == 4:
                break
            else:
                print("Invalidchoice.Pleasetry again.")
    def search_matches(self):
        sport = self.user_interface.get_sport()
        matches = self.database.find_matches(sport)
        if matches:
            self.user_interface.display_matches(matches)
        else:
            print("Nomatches found.")
    def view_upcoming_matches(self):
        location = self.user_interface.get_location()
        upcoming_matches = self.database.get_upcoming_matches(location)
        if upcoming_matches:
            self.user_interface.display_matches(upcoming_matches)
        else:
            print("Noupcomingmatches found.")
    def connect_with_participants(self):
        match_id = self.user_interface.get_match_id()
        participants = self.database.get_participants(match_id)
        if participants:
            self.user_interface.display_participants(participants)
        else:
            print("Noparticipantsfound.")
```

Figure 33: Codes generated by MACNET, part 2.

Multi-Agent Generated Software

user_interface.py

```
'''
User Interface module for the Sports application. The `UserInterface`
    class handles user interactions, displaying a menu for searching
    matches, viewing upcoming matches, and connecting with participants.
    It processes user inputs and displays match and participant details.
'''
class UserInterface:
    def display_menu(self):
        print("Welcometo the Sports application!")
        print("1.Search Matches")
        print("2.View UpcomingMatches")
        print("3.Connectwith Participants")
        print("4.Exit")
    def get_choice(self):
        while True:
            try:
                choice = int(input("Enteryourchoice:"))
                if choice in [1, 2, 3, 4]:
                    return choice
                else:
                    print("Invalidchoice.Pleaseenter a number between1 and 4."
                        )
            except ValueError:
                print("Invalidinput.Pleaseenter a number.")
    def get_sport(self):
        return input("Enterthesportyou are interestedin: ")
    def get_location(self):
        return input("Enteryourlocation:")
    def get_match_id(self):
        return int(input("EnterthematchID: "))
    def display_matches(self, matches):
        print("Matches:")
        for match in matches:
            print(match)
    def display_participants(self, participants):
        print("Participants:")
        for participant in participants:
            print(participant)
```

main.py

```
'''
This is the main file of the Sports application.
'''
from sports_app import SportsApp
def main():
    app = SportsApp()
    app.run()
if __name__ == "__main__":
    main()
```

Figure 34: Codes generated by MACNET, part 3.

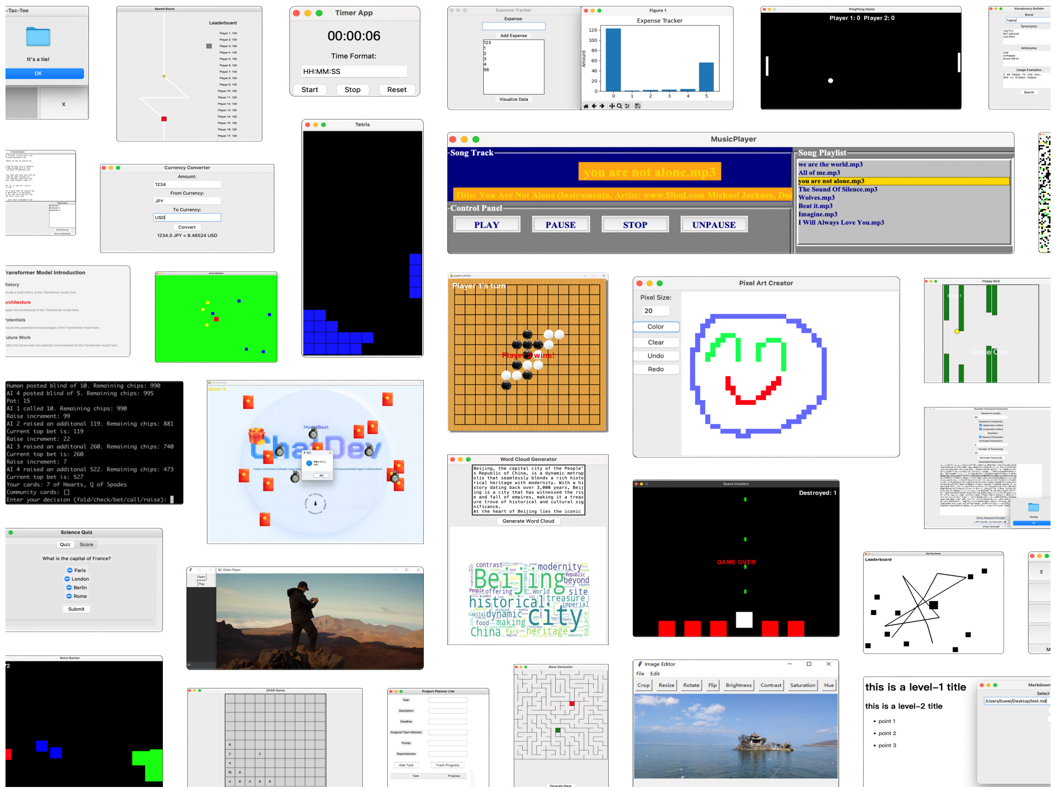


Figure 35: The software repository crafted by MACNET-CHAIN encompasses a diverse array of software categories, including but not limited to the game category and tool category. Each category contains a range of applications, each uniquely designed to meet specific user requirements and functionalities. The game category includes a variety of games developed using MACNET-CHAIN, ranging from simple puzzle games to more complex strategy and simulation games. These games are designed not only for entertainment but also to demonstrate the capabilities of MACNET-CHAIN in handling intricate logic, graphics, and user interaction. The tool category comprises various utility and productivity tools. Examples might include applications for data analysis, task management, or content creation. These tools are tailored to enhance productivity and efficiency, showcasing MACNET-CHAIN's ability to create software that addresses practical, everyday needs. In addition to these categories, the MACNET-CHAIN-created software warehouse likely includes many other types of software, each illustrating the versatility and breadth of applications that can be developed using this advanced development platform.