Multi-Agent Software Development through Cross-Team Collaboration

Zhuoyun Du[†♣] **Chen Qian**[†★] **Wei Liu**[★] **Zihao Xie**[★]
Yifei Wang[★] **Yufan Dang**[★] **Weize Chen**[★] **Cheng Yang**[♠✉]
[♣]Zhejiang University [★]Tsinghua University
[♠]Beijing University of Posts and Telecommunications
duzy.zju@outlook.com qianc62@gmail.com yangcheng@bupt.edu.cn

Abstract

The latest breakthroughs in Large Language Models (LLMs), *e.g.,* ChatDev, have catalyzed profound transformations, particularly through multi-agent collaboration for software development. LLM agents can collaborate in teams like humans, and follow the waterfall model to sequentially work on requirements analysis, development, review, testing, and other phases to perform autonomous software generation. However, for an agent team, each phase in a single development process yields only one possible outcome. This results in the completion of only one development chain, thereby losing the opportunity to explore multiple potential decision paths within the solution space. Consequently, this may lead to obtaining suboptimal results. To address this challenge, we introduce ***C**ross-**T**eam **C**ollaboration* (CTC), a scalable multi-team framework that enables orchestrated teams to jointly propose various decisions and communicate with their insights in a cross-team collaboration environment for superior content generation. Experimental results in software development reveal a notable increase in quality compared to state-of-the-art baselines, underscoring the efficacy of our framework. The significant improvements in story generation demonstrate the promising generalization ability of our framework across various domains. We anticipate that our work will guide LLM agents towards a cross-team paradigm and contribute to their significant growth in but not limited to software development. The code and data will be available at https://github.com/OpenBMB/ChatDev.

1 Introduction

In the field of software development, the complexity is profound, requiring a synergistic effort from professionals with a spectrum of expertise (Basili, 1989; Sawyer and Guinan, 1998). This collaborative process involves extensive communication to interpret and analyze project requirements through natural language, complemented by the development and debugging phases executed in programming languages (Ernst, 2017; Banker et al., 1998). Recent advancements in deep learning techniques have motivated researchers to explore their application in software engineering, to enhance effectiveness, and efficiency while reducing costs (Ezzini et al., 2022; López-Martín and Abran, 2015). Prior studies in deep learning-based software engineering have addressed a variety of tasks, including phases in the development chain like software requirements, design, implementation, testing, and maintenance (Pudlitz et al., 2019; Nijkamp et al., 2022). However, the methods used in these different phases have typically been isolated due to inconsistencies in their application.

The rapid advancement of Large Language Models (LLMs) has yielded remarkable achievements across various domains like natural language processing (Vaswani et al., 2017; Brown et al., 2020), text generation (Bubeck et al., 2023) and programming (Richards, 2023; Dong et al., 2023). However, limitations like hallucinations inherent in their standalone capabilities (Richards, 2023; Agnihotri and Chug, 2020), impede LLM's ability to generate usable content for task solving when confronted with complexities surpassing mere chatting. Recent progress in *autonomous agents*, with the integration of sophisticated features like context-sensitive memory (Park et al., 2023), multi-step planning (Wei et al., 2022b), and effective utilization of external tools (Schick et al., 2024), has enhanced their collaborative abilities. Through linguistic interaction, different agents can now effectively tackle a broader range of complex tasks include but is not limited to, mathematical reasoning (Wei et al., 2022b; Lu et al., 2022), software development (Osika, 2023; Qian et al., 2024c),

[†]Equal Contribution.
[✉]Corresponding Author.

game playing (Wang et al., 2023a; Zhu et al., 2023; Wang et al., 2023c; Gong et al., 2023), social simulation (Park et al., 2023; Li et al., 2023b; Zhou et al., 2023b; Hua et al., 2023), and scientific research (Huang et al., 2023; Liang et al., 2023).

When tackling complex tasks, great performance often necessitates collaboration (Hardin, 1968; Rand and Nowak, 2013; Hutter et al., 2011; Wang et al., 2023e; Woolley et al., 2010; Fehr and Gächter, 2000; Chen et al., 2023b; Kaddour et al., 2023). A noteworthy breakthrough in collaborative autonomous agents lies in the integration of interactions among multiple agents (Park et al., 2023; Li et al., 2023a; Qian et al., 2024c,a,b). Typical methods (Qian et al., 2024c; Hong et al., 2023) decompose task into several distinct subtasks. An instructor agent and an assistant agent are assigned to solve each subtask. The instructor gives instructions on the subtask and the assistant responds with a solution. Through multi-turn autonomous communication between agents, they collaboratively generate content (e.g., software, math results, scientific conclusion) for the task. It is noteworthy that the content produced can vary across multiple iterations given the same task, reflecting the dynamic nature of the problem-solving process (Achiam et al., 2023). In the field of multi-agent systems for software development, a series of autonomous agents interact through a development chain with multiple configurable task-oriented phases can be regarded as a single-team. The team completes the development chain through chat chain (Qian et al., 2024c) and generates the sequential, task-oriented data (such as requirement documents, codes, test cases, and user manuals), which can be regarded as a decision path within the solution space.

However, one agent team can only execute all phases sequentially according to its pre-defined team configuration (e.g., the number of agents, agent profiles, LLM hyperparameters), and its decision path is fixed (Qian et al., 2024c; Hong et al., 2023; Qian et al., 2024a,b). This design may lead to repetitive errors by an agent team with a particular configuration when facing a certain type of problem, preventing self-correction. Additionally, it restricts the agents from exploring more and better decision paths. Therefore, it is necessary to introduce multiple agent teams that are aware of each other, enabling them to collaborate effectively to explore more potential paths. Then the challenge becomes: *How can multi-agent systems obtain and utilize insights from others to achieve a superior outcome?* In this paper, we propose ***C**ross-**T**eam **C**ooperation* (CTC), a framework that carefully orchestrates agents into multiple teams, each with the same task assignment to communicate in a cooperative environment. Specifically, our framework enables different teams to concurrently propose various task-oriented decisions as insights for content generation (single-team proposal) and then communicate for insights interchange in some important phases (multi-team aggregation). Different agent teams utilize a greedy pruning mechanism to eliminate low-quality content and then carry out a solution aggregation mechanism to aggregate various content into a superior outcome collaboratively.

Through our experiments with 15 tasks from different categories and styles randomly selected from the SRDD dataset (Qian et al., 2024c) for software generation (programming-language-oriented reasoning), we demonstrate a significant improvement in software quality using the proposed framework. We highlight the importance of diversity across teams and emphasize the importance of fostering a cross-team collaboration environment in bolstering teams' performance through our pruning mechanism. Furthermore, to further demonstrate the generalizability of our framework, we extended its application to the domain of story generation (natural-language-oriented reasoning), randomly incorporating 10 tasks from the ROCStories dataset (Chen et al., 2019). The results revealed a notable improvement in story quality. Our findings underscore the efficacy and promising generalization of our framework in complex tasks. In summary, our contributions are threefold:

- We propose Cross-Team Collaboration (CTC), a scalable multi-team collaboration framework that efficiently orchestrates LLM agents into multiple teams to perform multi-team interactions, which facilitates seamless content exchange among agent teams and effectively supports the generation of diverse content forms, including programming language and natural language.

- Our approach involves concurrent reasoning within each team, followed by the aggregation of diverse content from multiple teams into a superior outcome through greedy pruning, which effectively incorporates multidimensional solutions by retaining their strengths and eliminating their drawbacks.

- We conducted extensive experiments demonstrating the effectiveness and generalizability of our framework in software development, indicating that multi-team collaboration outperforms individual efforts.

Organization. The subsequent sections of this paper is organized as follows. We highlight some works related to this paper in Section 2. Section 3 provides preliminaries of our work. Then we provide the methodology of our framework in Section 4, detailing the formation of a single agent team and the method to orchestrate these teams. Section 5 describes the dataset used in our experiment and shows the experimental results and analysis. The limitations of our work are discussed in Section 6. Finally, we conclude the paper in Section 7, and suggest potential directions for future research.

2 Related Work

Software engineering (SE) is the systematic, rigorous, and measurable process of designing, developing, testing, and maintaining software[1]. The complexity inherent in SE often necessitates decision-making that is heavily reliant on intuition and, in the best-case scenario, consultation with experienced developers. Previous research on deep learning (DL) has shown remarkable promise when applied to SE (Pudlitz et al., 2019; López-Martín and Abran, 2015; Alahmadi et al., 2020; Wang et al., 2021; Wan et al., 2022, 2018; Nahar et al., 2022). The emergence of LLMs has yielded novel solutions. Trained on vast datasets with extensive parameters, LLMs have revolutionized the landscape of natural language processing (Brown et al., 2020; Bubeck et al., 2023; Vaswani et al., 2017; Radford et al., 2019; Touvron et al., 2023; Wei et al., 2022a; Shanahan et al., 2023; Chen et al., 2021; Brants et al., 2007; Ouyang et al., 2022; Yang et al., 2023; Qin et al., 2023b; Kaplan et al., 2020; Achiam et al., 2023). Their impact is particularly significant within the realm of autonomous agents (Zhou et al., 2023a; Wang et al., 2023a; Park et al., 2023; Wang et al., 2023c; Richards, 2023; Osika, 2023; Wang et al., 2023d), where these agents exhibit proficiency in task decomposition planning (Chen et al., 2023b; Liu et al., 2023), retrieval-augmented memory (Park et al., 2023; Sumers et al., 2023), and strategic tool utilization (Schick et al., 2024; Cai et al., 2023; Qin et al., 2023a; Ruan et al., 2023; Yang et al., 2024), thus enabling independent operation within intricate real-world contexts (Zhao et al., 2024; Zhou et al., 2023a; Ma et al., 2023; Zhang et al., 2023; Wang et al., 2023b; Ding et al., 2023; Weng, 2023). Additionally, LLMs have exhibited formidable role-playing capabilities in various designated roles (Li et al., 2023a; Park et al., 2023; Hua et al., 2023; Chan et al., 2023; Zhou et al., 2023b; Chen et al., 2023b; Cohen et al., 2023; Li et al., 2023b). Recent exploration of autonomous interactions among multiple agents heralds a promising paradigm shift towards collaborative multi-agent task-solving (Li et al., 2023a; Qian et al., 2024a; Park et al., 2023; Zhou et al., 2023b; Chen et al., 2023b; Chan et al., 2023; Chen et al., 2023a; Cohen et al., 2023; Li et al., 2023b; Hua et al., 2023). These systems, which assign distinct roles to an instructor and an assistant, foster collaborative interactions among autonomous agents. This collaboration efficiently decomposes complex tasks into manageable subtasks (Qian et al., 2024c; Hong et al., 2023; Wu et al., 2023). The instructor provides directional instructions, while the assistant offers pertinent responses. This approach not only enhances productivity but also facilitates well-orchestrated workflow for task completion, thereby significantly reducing the necessity for human intervention (Li et al., 2023a; Qian et al., 2024c; Chen et al., 2023b). An exemplary instance is ChatDev (Qian et al., 2024c), a virtual software company powered by LLMs. It leverages agents in roles such as reviewer and programmer within a chat-chain workflow, breaking the isolation of steps in the software engineering (SE) process and substantially enhancing the efficiency of software development.

Recent studies have highlighted the significance of multi-agent collaboration and competition in enhancing their performance in scenarios like programming, game playing, and reasoning (Wang et al., 2024; Light et al., 2023; Li et al., 2024; Duan et al., 2024; Xu et al., 2023; Piatti et al., 2024; Dong et al., 2023). Additionally, research on scaling the number of agents pointed out that the use of simple voting mechanisms and straightforward topological communication strategies can enhance the quality of content generated by multiple agents (Li et al., 2024; Yin et al., 2023). Research on Graph-like multi-agent systems has introduced a novel perspective on the structuring of communication networks (Hu et al., 2023; Zhuge et al., 2024; Besta

[1] www.computer.org/sevocab

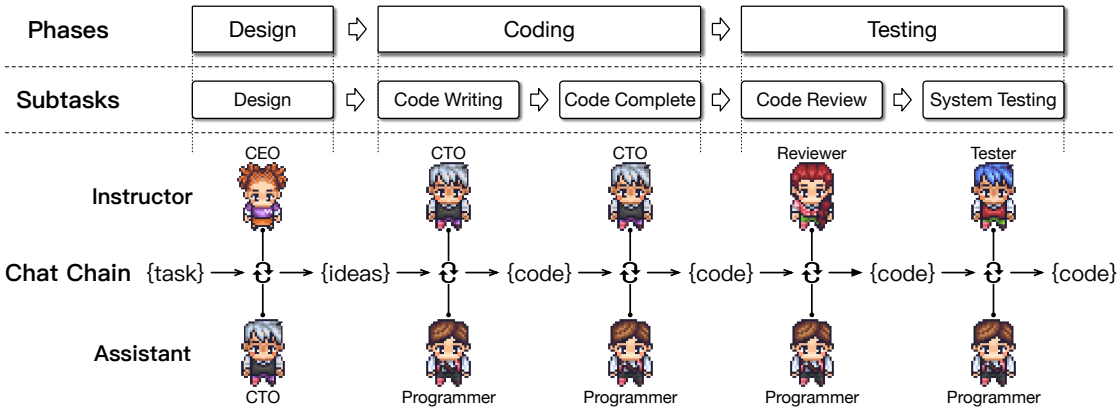

Figure 1: A single-team integrates LLM agents assigned to various roles such as requirements analysts, professional programmers, and test engineers. Upon reception of an initial task requirement (e.g., *"Develop a Tetris game"*), the agents engage in multi-turn communication and sequentially complete subtasks along a chat chain. Through autonomous collaboration on a series of subtasks, they collectively devise a comprehensive solution.

et al., 2024; Jiang et al., 2023), endowing the communication architecture with enhanced dynamism and facilitating more efficacious content exchange among the constituent agents. Nonetheless, an increase in agent calls may engender diminishing returns, and in some instances, may even degrade model performance (Chen et al., 2024; Piatti et al., 2024). Suggesting that it is not viable for further improvement in the performance of a multi-agent system by solely increasing the scale of the agents.

3 Preliminaries

Before delving into the main idea, we introduce some background knowledge: Chain as Team and Agent Communication (Qian et al., 2024c; Li et al., 2023a). These concepts serve as the foundational components of our proposed architecture.

Definition 1 (Chain as Team) *As illustrated in Figure 1, a single-team ($\mathcal{C}$) is conceptualized as a chain-like structure composed of a series of task-oriented phases ($\langle \mathcal{P}^1, \mathcal{P}^2, \ldots, \mathcal{P}^{|\mathcal{C}|} \rangle$) that sequentially address the resolution of tasks which can be formulated as:*

$$\mathcal{C} = \langle \mathcal{P}^1, \mathcal{P}^2, \ldots, \mathcal{P}^{|\mathcal{C}|} \rangle \tag{1}$$

We refer to such a chain-like structure as a team that could participate in our CTC framework.

Definition 2 (Agent Communication) *In each phase, agent communications occur between the Instructor ($\mathcal{I}$) and the Assistant ($\mathcal{A}$), adhering to a straightforward instruction-response format, as depicted:*

$$\langle \mathcal{I} \rightarrow \mathcal{A}, \quad \mathcal{A} \leftarrow \mathcal{I} \rangle_{\circlearrowleft} \tag{2}$$

To mitigate communication hallucinations, a "role reversal" mechanism is implemented. In this mechanism, the assistant assumes an instructor-like role, actively seeking more detailed information (e.g., the exact name of an external dependency and its associated class) before providing a conclusive response. Once the instructor offers a specific modification suggestion, the assistant then carries out precise optimization, as illustrated:

$$\langle \mathcal{I} \rightarrow \mathcal{A}, \ \langle \mathcal{A} \rightarrow \mathcal{I}, \ \mathcal{I} \rightsquigarrow \mathcal{A} \rangle_{\circlearrowleft}, \ \mathcal{A} \rightsquigarrow \mathcal{I} \rangle_{\circlearrowleft} \tag{3}$$

This mechanism addresses one specific issue at a time, necessitating multiple rounds ($\circlearrowleft$) of communication to refine and optimize various potential problems.

4 Methodology

In this section, we focus on studying collaborative behavior among multi-teams that share with same task objective. In the single-team that performs intra-team collaboration (*Single-Team Execution*), a preliminary idea is given at the start, and agents arranged in a chain-like structured team will complete it autonomously through conversations. To transcend the isolated nature of Single-Team Execution and harness insights effectively from multiple teams, it is crucial to devise a framework that facilitates effective cross-team interactions, enabling teams to engage collaboratively, culminating in a consensus that yields superior content.

4.1 Single-Team Execution

The direct generation of complex content using LLMs could lead to hallucinations (Mann

et al., 2020). These hallucinations may manifest as incomplete content, misinformation, and non-compliant responses. The hallucinations primarily stem from two aspects. Firstly, the lack of guidance in tasks confuses LLMs when generating entire content at once (Azamfirei et al., 2023). Secondly, the absence of cross-examination in the content-generating process exposes considerable vulnerabilities. Individual model instances present a varied spectrum of responses, throwing the need to debate or examine the responses from other model instances to reach a unified and refined consensus (Du et al., 2023; Yin et al., 2023; Li et al., 2023a; Qian et al., 2024c; Chan et al., 2023; Du et al., 2023).

Single-Team Execution addresses the aforementioned issues to a certain extent by forming agents into a chain-like team that performs intra-team collaboration to divide the task into manageable subtasks and conquer them through phases various from decision making, and designing to content writing along the chain. Inspired by ChatDev (Qian et al., 2024c), a Single-Team Execution ($\mathcal{C}$) is composed of multiple phases ($\mathcal{P}$), each divided into atomic subtasks focused on role-playing with two roles: instructor ($\mathcal{I}$) and assistant ($\mathcal{A}$). The instructor initiates dialogues by providing instructions ($\rightarrow$) to guide the subtask ($\mathcal{T}$), while the assistant follows these instructions and responds with ($\rightsquigarrow$) solutions. Through a multi-turn dialogue (C), they collaborate to reach a consensus, extracting (τ) solutions ranging from text to code, thus completing the subtask. The comprehensive process of the task-solving process along the chat chain can be formulated as:

$$\begin{aligned}
\mathcal{C} &= \langle \mathcal{P}^1, \mathcal{P}^2, \ldots, \mathcal{P}^{|\mathcal{C}|} \rangle \\
\mathcal{P}^i &= \langle \mathcal{T}^1, \mathcal{T}^2, \ldots, \mathcal{T}^{|\mathcal{P}^i|} \rangle \\
\mathcal{T}^j &= \tau\big(\mathsf{C}(\mathcal{I}, \mathcal{A})\big) \\
\mathsf{C}(\mathcal{I}, \mathcal{A}) &= \langle \mathcal{I} \rightarrow \mathcal{A},\ \mathcal{A} \rightsquigarrow \mathcal{I} \rangle_{\circlearrowleft}
\end{aligned} \tag{4}$$

A chat chain corresponds to a decision pathway in solution space, offering a transparent view of the content generation process by the agents in a team.

4.2 Cross-Team Collaboration

Facing diverse tasks, Single-Team Execution often tackles tasks in isolation. The quality and tendency of content produced by a Single-Team Execution in a phase are predominantly determined by the decisions made by prior phases, which are then "baton passed" to a subsequent phase. This process, while streamlined, lacks the diversity of insights necessary to explore a wider range of decision pathways for superior content. A straightforward approach is to run n teams simultaneously given the same task and ensemble the results from these n teams. However, this simple and direct ensembling of results overlooks the mutual awareness between teams during intermediate phases, leading to insufficient collaboration. This is akin to exploring n paths in parallel, whereas crossing the intermediate phase nodes of these paths can potentially explore more paths. Yet, this introduces new issues, such as a significant increase in communication bandwidth and the possibility of incorporating noise from underperforming teams. To address this, we propose a new cross-team cooperative framework. In this setup, intermediate phase nodes involve both intra-team and inter-team collaboration. Within each team, members work together harmoniously, while between teams, strengths from others are leveraged to compensate for weaknesses. Essentially, this approach allows for the exploration of more potential paths through intermediate node crossover while also pruning to ensure the quality of the candidate paths.

We propose Cross-Team Collaboration (CTC), which is denoted by $\mathcal{N}$. CTC orchestrates parallel executions of Single-Team Execution ($\mathcal{S} = \{\mathcal{C}^1, \mathcal{C}^2, \cdots, \mathcal{C}^n\}$) with configurable temperature and length of chains[2], and each team is assigned the same task objective. These teams jointly propose various task-oriented decisions based on their different perspectives. After key phases ($\mathcal{K}$) like designing and writing where important decisions or significant content changes are made, teams would "wait" and extract the contents for communications ($\mathcal{E} = \{e_1, e_2, \cdots, e_{|\mathcal{K}|}\}$) and facilitate a greedy pruning mechanism to eliminate low-quality content. After that contents are partitioned into groups and collaboratively aggregated into more superior content. This communicative dynamic can be naturally modeled as:

$$\mathcal{N} = \{\mathcal{C}^i \mid \mathcal{C}^i \in \mathcal{S}\} \cup \mathcal{E} \qquad \mathcal{E} = \{e_i \mid \mathcal{P}^i \in \mathcal{K}\} \tag{5}$$

Through Cross-Team Collaboration, a cross-team network is established, driving teams to be

[2]Length Diversity can be induced manually and autonomously. Even after the configuration of various phases for the teams, the length of the chat chains can continue to vary autonomously along the decision-making process.

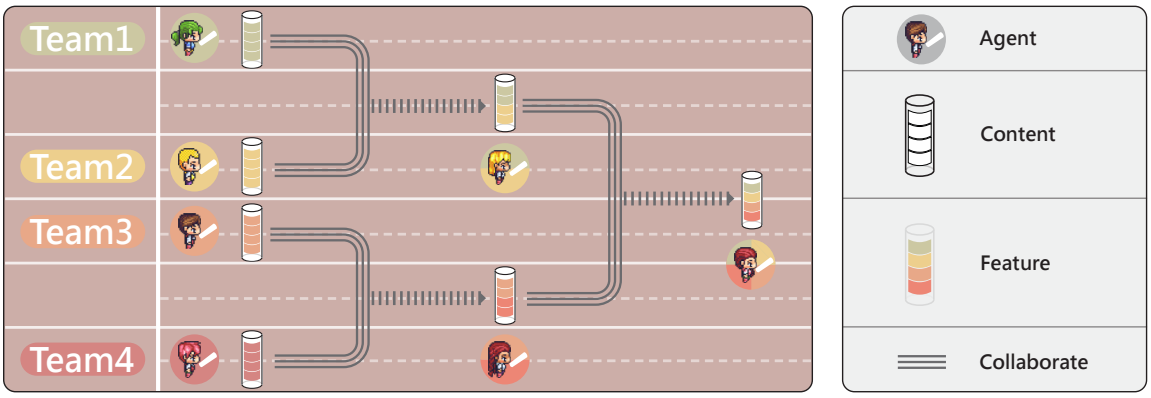

Figure 2: The aggregation process in Cross-Team Collaboration involves multiple agents ( ) from different teams contributing a variety of content ( ). These contents are partitioned into groups and cooperatively ( ) integrated through communications, highlighting the distinctive features ( ) of each team. Ultimately, this process results in a superior outcome that embodies the features of all participating teams.

more innovative and effective in the production of superior content.

4.2.1 Greedy Pruning

In the realm of real-world scenarios, the pursuit of optimal outcomes often necessitates the strategic placement of multiple teams within a competitive and collaborative environment (Hardin, 1968; Rand and Nowak, 2013; Hutter et al., 2011; Piatti et al., 2024; Chen et al., 2023b). As the number of teams initially configured in Cross-Team Collaboration increases, wider insights are proposed, causing a significant expansion in generating pathways of the final content. Meanwhile, we observe that not all content as insights that participate in communications contributes positively to the optimized content, low-quality content can harm the quality of content and increase the communication burden. Therefore, we introduce a *Greedy Pruning* mechanism to filter out some teams early on, allowing only the most promising ones to proceed to engage in inter-team collaboration. This process involves a rigorous quality assessment conducted on content, which results in the elimination of a predefined proportion of the content items with low quality from teams, greedily electing high-quality content for the following processes. This quality assessment and pruning mechanism are designed to balance the quantity and quality of content, ensuring that the most valuable contributions are carried forward while the communication burden is bearable. A comprehensive breakdown of the assessment methodology can be found in *Evaluation* 5 section, where the details and efficacy of our Greedy Pruning mechanism are thoroughly examined. After pruning a proportion of content, we propose to use a first-partition-then-aggregate process for the filtered content which makes diverse pathways eventually converge into a singular, cohesive outcome.

4.2.2 Hierarchy Partitioning.

To prevent long-context issues rooted in the overwhelming amount of simultaneous content communications burden on an agent, We propose a *Hierarchy Partitioning* mechanism where teams are divided into groups to collaborate and merge. The outcome of each group advances to the next level, continuing the process until only one team is left. Through uniform partitioning with expected quantity u of content per communication, we obtain a set of *communicative groups* $\mathcal{G} = \{g_1, g_2, \ldots, g_{\frac{n}{u}}\}$. Each element g_i comprises a subset of content $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$ participates in collaboration. After partitioning, the aggregation mechanism ($\rightarrowtail$) is conducted, generating an optimized set of content $\mathcal{R}^1 = \{r_1^1, r_2^1, \ldots, r_{|G|}^1\}$. These optimized content undergo a subsequent partitioning into new groups. This process can be formalized as follows:

$$\begin{aligned} \mathcal{G}^{k+1} &= \left\{ g_i^{k+1} \mid g_i^{k+1} \subseteq \mathcal{R}^k \right\} \\ \mathcal{G}^{k+1} &\rightarrowtail \mathcal{R}^{k+1} \end{aligned} \tag{6}$$

where $\mathcal{G}^k$ is the set of communicative groups and $\mathcal{R}^k$ is the set of contents that participates in communications at aggregate iteration k. This iterative process persists until the cardinality of $\mathcal{R}$ is one, which is the final aggregated content, leading to the formation of hierarchical communicative groups.

4.2.3 Greedy Aggregation.

In *Hierarchy Partitioning*, the teams within each group need to collaborate and determine the best in the group. This process is not simply about eliminating teams to select the best one, but rather about combining the strengths of all teams while eliminating their weaknesses, as shown in Figure 2. Essentially, it is a process of synthesizing multiple decision paths into a single, optimal path. To aggregate various pathways into a superior one, we introduce an aggregation mechanism harnessing the features of content. In one *Greedy Aggregation* process, a role-assigned agent $\mathcal{M}$ meticulously extracts the strengths and drawbacks of each content within each communicative group. It then collaboratively aggregates ($\rightarrowtail$) a rewritten content that greedily integrates strengths and eliminates drawbacks. Subsequently, outlines the changes made in the rewritten content explicitly. This comprehensive report facilitates a better understanding of the rewritten content and supports further optimization and improvement efforts, boosting the performance of $\mathcal{M}$. This aggregation process can be represented as:

$$\begin{cases} \mathcal{M}(g_i^k) \rightarrowtail r_i^{k+1} & |g_i^k| \neq 1 \\ g_i^k \rightarrow r_i^{k+1} & |g_i^k| = 1 \end{cases} \tag{7}$$

Where r represent a rewritten content from an agent, and g_i^k represents a communicative group comprising $\{r_j^k, r_{j+1}^k, \ldots, r_{j+u}^k\}$. If a communicative group contains only a single piece of content, it is directly transferred ($\rightarrow$) to become the rewritten content without undergoing the aggregation process.

5 Evaluation

In this section, we scrutinize the intriguing observations, challenging issues, and several examples encountered in the experiments of Cross-Team Collaboration. We also give detailed analysis across different hyperparameters and settings like team sizes, temperatures, and mechanism configurations.

5.1 Overall Performance

Experiment Setup In our experiments, we employ GPT-3.5-Turbo as the foundational model, utilizing the Cross-Team Collaboration framework described in Section 4.2. We limit communication rounds between agents to a maximum of 5 per phase in each Single-Team Execution. By default, the number of teams engaged in the tasks is set to 8 and the temperature parameter is 0.2. We conduct a Greedy Pruning mechanism only on 8-team CTC in our experiments. We introduce a waiting phase for the Cross-Team Collaboration after the *coding* and *code completion* phases[3] for code generation tasks and after the *writing* phase for story generation tasks. Our software generation experiments randomly draw 15 tasks from the SRDD dataset (Qian et al., 2024c), a specialized open-source collection tailored for the "Natural Language to Software Generation" domain, and 10 tasks for story generation from ROCStories (Chen et al., 2019), a collection of commonsense 5 sentences short stories can be used for longer stories generation. In establishing our baselines, we compare against GPT-Engineer (Osika, 2023), a single-agent approach to software development, as well as ChatDev (Qian et al., 2024c), MetaGPT (Hong et al., 2023), and AgentVerse (Chen et al., 2023b), which represent the state-of-the-art multi-agent single-team paradigms for software development. We also incorporate GPTSwarm (Zhuge et al., 2024), a graph-like multi-agent single-team agent framework as a strong baseline. The performance metrics are the average across all tasks within the test set. All baseline evaluations adhere to our proposed framework's same hyperparameters and settings to ensure a fair comparison.

Metrics for Software Evaluation Evaluating software generated by LLM is a challenging task, especially when assessing it on a holistic level. As a solution, we use four fundamental dimensions to assess specific aspects of the software proposed by previous works (Qian et al., 2024a,c).

- *Completeness*[4] ($\alpha \in [0, 1]$) measures the software's capacity for comprehensive code fulfillment during development. It is measured by the proportion of the software that is free from "TODO"-like placeholders. A higher score implies a greater likelihood of the software being

[3] A Single-Team Execution in CTC mainly comprises demand analysis, coding, code completion, reviewing, and testing phases. The coding phase involves a single round of agents' cooperative communication, while the code completion, reviewing, and testing phases each entail multiple rounds.

[4] Significantly different from traditional function-level code generation, a prevalent observation in agents' software development is the frequent use of numerous "placeholder" fragments, such as Python's `pass` statement. This practice, which indicates a significant level of incompleteness within the software, is a problem infrequently noted in previous work. Given these challenges, completeness should be regarded as a primary metric for evaluating the quality of software generated by agents.

Method	Paradigm	Completeness	Executability	Consistency	Quality
GPT-Engineer		$0.502^{\dagger}$	$0.358^{\dagger}$	$0.768^{\dagger}$	$0.543^{\dagger}$
MetaGPT		$0.483^{\dagger}$	$0.415^{\dagger}$	$0.739^{\dagger}$	$0.545^{\dagger}$
ChatDev		$\underline{0.744}^{\dagger}$	$0.813^{\dagger}$	$\underline{0.781}^{\dagger}$	$\underline{0.779}^{\dagger}$
AgentVerse		$0.650^{\dagger}$	$\underline{0.850}^{\dagger}$	$0.776^{\dagger}$	$0.759^{\dagger}$
GPTSwarm		**0.800**	$0.550^{\dagger}$	$0.779^{\dagger}$	$0.710^{\dagger}$
CTC		$\underline{0.795}$	**0.928**	**0.796**	**0.840**

Table 1: Overall performance comparison of various representative software development methods, encompassing single-agent( ), Single-Team Execution ( ), Graph-like Execution ( ) and Cross-Team Collaboration ( ) framework. The performance metrics are the average across all tasks within the test set. The highest scores are highlighted in **bold**, and the second-highest scores are presented with underline. † indicates significant statistical differences ($p < 0.05$) between baselines and ours.

capable of automated completion without the need for further manual coding.

- *Executability* ($\beta \in [0, 1]$) assesses the software's ability to run correctly within a given compilation environment. It is measured by the percentage of software that compiles without errors and is ready to execute. A higher score indicates a higher likelihood of the software running successfully as intended.

- *Consistency* ($\gamma \in [0, 1]$) evaluates the alignment between the generated software and the original natural language requirements. It is quantified as the cosine distance between the embeddings of the text requirements and the source code. A higher score indicates a greater degree of compliance with the requirements.

- *Quality* ($\frac{\alpha+\beta+\gamma}{3} \in [0, 1]$) is a comprehensive metric that integrates the dimensions of completeness, executability, and consistency. It serves as a holistic indicator of the software's overall quality. A higher score indicates superior generation quality, suggesting that the software is less likely to require additional manual interventions.

Table 1 illustrates a detailed comparative analysis of our Cross-Team Collaboration framework (CTC) and all baselines. The Single-Team Execution paradigm outperforms the GPT-Engineer in terms of overall performance, highlighting the benefits of a multi-agent system in decomposing complex task-solving into manageable subtasks, as opposed to a single-step solution approach. Across all metrics, CTC demonstrates a remarkable improvement over the Single-Team Execution, showing only a slightly lower score in Completeness when compared to the Graph-like Execution paradigm but significantly higher in Executability. The contrast with ChatDev, an powerful multi-agent framework, is especially noteworthy, the Completeness score escalates from 0.744 to 0.795, and the Executability score witnesses a substantial leap from 0.813 to 0.928, and the Consistency score improves from 0.781 to 0.796, the overall quality of the generated software significantly improves from 0.779 to 0.840. These enhancements underscore the advantages of the CTC framework, where collaborations among teams lead to mutual correction and enlightenment, subsequent enhancement in software quality, reducing the likelihood of executable errors, and elevating the degree of code completion and alignment with user requirements.

5.2 Hyperparameter Analysis

The Number of Teams Our investigation, as delineated in Figure 3, uncovers an intriguing inverse relationship between the executability and completeness of the software generated by our framework. This figure succinctly captures the essence of the trade-off that is inherent in the system's performance. Initially, we observe an ascent in the alignment of generated code with specified requirements, plateauing around the 4-team CTC configuration, with minor fluctuations as the number of teams increases. The zenith of software quality is achieved with the 4-team CTC configuration. This configuration strikes a delicate balance, optimizing the system to produce software that is not only executable but also functionally rich. However, upon further increasing the number of teams, we observe a decline in the quality of the generated software. Despite this decrement, it is noteworthy that the software's quality remains superior to that of the baseline Single-Team Execution configuration. We

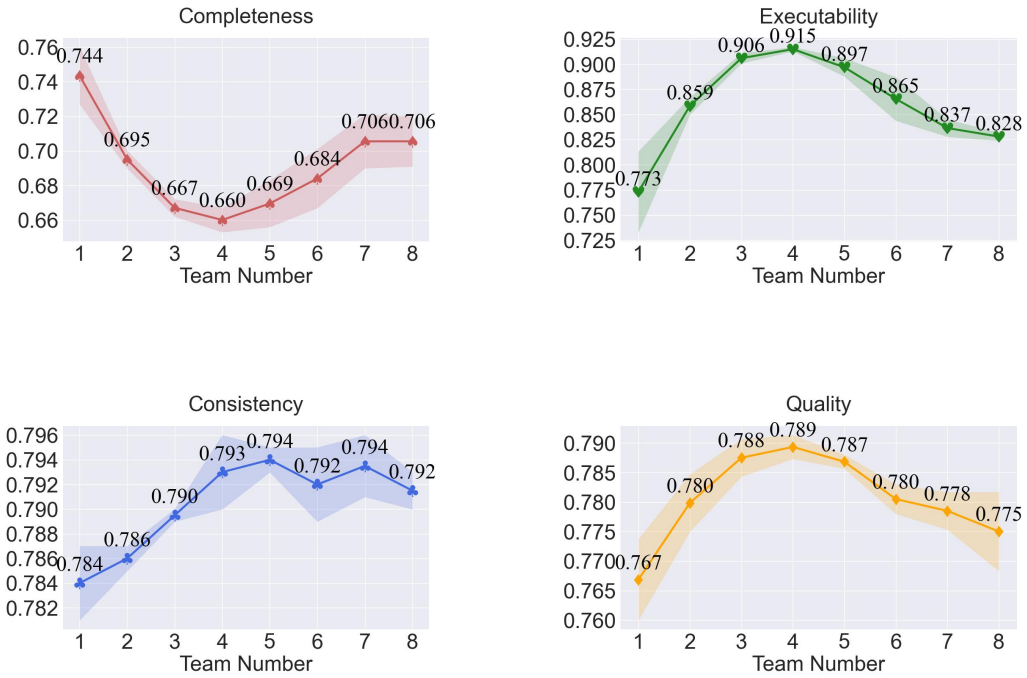

Figure 3: Visualization of Result Trends concerning Team Size Variations in our Framework without Greedy Pruning. An upward trend in consistency is observed, along with an inverse relationship between executability and completeness. The highest quality of content is achieved with a team size of four.

hypothesize that the diminishing returns and potential decline in performance are due to the agents' inability to process an excessive volume of content simultaneously. To further enhance the number of teams without compromising quality, the implementation of a greedy pruning mechanism becomes indispensable.

Temperature A central focus of our investigation is the efficacy of varied temperature configurations for teams to generate content with different inclinations in creativity and requirement compliance. Table 2 demonstrates that an appropriate level of diversity significantly enhances software quality. When the temperature for each team is set to the same level, the performance improvement brought by CTC is limited. This is because there is no differentiation among the teams; they either all lean towards generating creative content (high temperature) or strictly adhering to rules (low temperature). In this scenario, the new information gained through cross-team collaboration is minimal. Conversely, when each team is assigned a different temperature (such as simply employing a temperature configuration of (0.2, 0.2, 0.4, 0.4), which balances team diversity with a focus on rules), CTC results in significant performance enhancement.

Greedy Pruning To enhance the scalability and performance of the Cross-Team Collaboration framework, the Greedy Pruning mechanism detailed in Section 4.2.1 is essential. The Greedy Pruning mechanism is applied to reduce communication costs and improve content quality for aggregations between teams. As indicated in Table 2, the application of the Greedy Pruning mechanism within the 8-team CTC context results in an optimization of all metrics dimensions, achieving the highest values across all CTC experimental outcomes. Thus its' quality scores exhibit a remarkable improvement, rising from a previous peak of 0.789 in the 4-team CTC to a more refined 0.840. These results show that the mechanism can handle the challenges of larger teams. It selects high-quality content to improve overall performance by optimizing the generation process. This makes it more useful and effective for software tasks. It supports the idea of using the Greedy Pruning mechanism in Cross-Team Collaboration to support scalable applications.

5.3 Ablation Study

In our ablation study illustrated in Table 3, we found that removing Hierarchical Partitioning from 4-team and 8-team CTC configurations reduced quality scores from 0.789 to 0.756 and from 0.775

Mechanism	Completeness	Executability	Consistency	Quality
4-team CTC	0.660	0.915†	**0.793**	0.789†
(0.2 0.4 0.6 0.8)	0.575	0.875†	0.790	0.747†
(0.1 0.1 0.1 0.1)	**0.700**	0.794†	0.791	0.762†
(0.4 0.4 0.4 0.4)	0.583	0.773†	0.792	0.716†
(0.2 0.2 0.4 0.4)	0.670	**0.925**	0.790	**0.795**
8-team CTC	0.706†	0.828†	0.792†	0.775†
+ Prune	**0.795**	**0.928**	**0.796**	**0.840**

Table 2: Investigation of mechanisms in 4-team and 8-team CTC. The temperatures for each team are indicated as (t_1, t_2, t_3, t_4). The '+' symbol represents the adding operation. The highest scores are highlighted in **bold**, and the second-highest scores are presented with underline. † indicates significant statistical differences ($p < 0.05$) between the scenarios with and without the addition of the mechanism.

Mechanism	Completeness	Executability	Consistency	Quality
4-team CTC	0.660	**0.915**	**0.793**	**0.789**
- Hierarchy Partitioning	**0.683**	0.800	0.786	0.756
- Feature Extraction	0.680	0.783	0.739	0.735
8-team CTC	0.706	**0.828**	**0.791**	**0.775**
- Hierarchy Partitioning	**0.728**	0.804	0.787	0.773
- Feature Extraction	0.658	0.783	0.790	0.744

Table 3: Ablation study on 4 Teams CTC and 8 Teams CTC. The '-' denotes the removing operation. The highest scores are highlighted in **bold**, and the second-highest scores are presented with underline in each subtable.

to 0.773. Without this partitioning, the agent struggled to handle diverse team content in one aggregation, making it harder to extract features and lowering content quality. Besides, we conducted ablation experiments on Content Feature Extraction, where agents do not have a role assignment, and there is no feature extraction performed by the assessment agent $\mathcal{M}$. The performance further dropped the quality from 0.789 to 0.735 and from 0.775 to 0.744. This ablation makes CTC perform poorly and sometimes even fail to complete tasks. The lack of structured guidance and groupings led to disorganized content and poor task resolution, with rewritten content quality dropping. In some cases, the agent would respond: "I cannot process this task." These results show the importance of our framework's mechanisms in managing complex content and ensuring high-quality outputs in multi-team scenarios.

5.4 Generalizability Analysis of Story Generation

To demonstrate the generalization capability of our framework, we have conducted experiments in the domain of story generation. Our findings indicate that our framework significantly enhances the quality of stories generated by both individual agents and Single-Team Execution. This improvement underscores the versatility and robustness of our approach across different domains.

Metrics for Story Evaluation Inspired by (Li et al., 2018), we evaluate story quality across four critical dimensions by using an LLM to rate each story, which is proven to be effective (Chhun et al., 2024).

- *Grammar and Fluency* ($\omega \in [0, 4]$): Assesses natural language use, grammatical correctness, and fluency for a coherent and error-free narrative flow.
- *Context Relevance* ($\psi \in [0, 4]$): Analyzes the contextual appropriateness and interrelation of names, pronouns, and phrases to ensure narrative integrity and depth in plots.
- *Logic Consistency* ($\xi \in [0, 4]$): Examines the logical progression of events and character relationships for narrative coherence and plausibility.
- *Quality* ($\frac{\omega+\psi+\xi}{3} \in [0, 4]$): Aggregates individual dimension scores to provide a comprehensive measure of narrative quality, reflecting the synthesis of language, context, and logic.

Team Number Analysis In our experimental investigation using the CTC framework for story generation, as depicted in Table 4, we observed a positive correlation between the number of participating teams and the resultant quality of the generated stories. Notably, the quality metrics demonstrated a substantial improvement over outputs from individual agents and Single-Team Execution setups, with scores rising from 2.193 and 2.358 to 3.083, respectively. However, as the number of teams increased, diminishing returns began to set in. To counteract this trend, we introduced the Greedy Pruning mechanism. This intervention led to a notable enhancement in story quality when the number of teams was eight, with the quality score improving from 3.083 to 3.642. These findings underscore the efficacy of the CTC framework in story generation, suggesting that it is not only beneficial for software development tasks but also generalizes well to creative domains such as narrative generation.

Ablation Study Similar to the ablation study in software generation, we conduct experiments regarding the impact of Hierarchical Partitioning and

Mechanism	Paradigm	Grammar and Fluency	Context Relevance	Logic Consistency	Quality
Single-Agent	😎	2.150†	2.005†	2.425†	2.193†
Single-Team Execution	🏳	2.250†	2.325†	2.500†	2.358†
2-team CTC	🏳🏳	2.725	2.800	3.000	2.842
3-team CTC	🏳🏳	2.967	2.767	2.967	2.900
4-team CTC	🏳🏳	2.967	2.850	2.908	2.908
5-team CTC	🏳🏳	2.980	2.880	2.960	2.940
6-team CTC	🏳🏳	2.983	2.900	2.983	2.956
7-team CTC	🏳🏳	3.000	3.171	3.014	3.062
8-team CTC	🏳🏳	3.000†	3.250†	3.000†	3.083†
8-team CTC + Prune	🏳🏳✂	**3.625**	**3.750**	**3.250**	**3.642**

Table 4: Result Trends concerning Team Size Variations in our Framework in Story Generation, encompassing single-agent(😎), Single-Team Execution (🏳) and Cross-Team Collaboration (🏳🏳) framework with and without pruning mechanism (✂). The performance metrics are the average across all tasks within the test set. The highest scores are highlighted in **bold**, and the second-highest scores are presented with underline. † indicates significant statistical differences ($p < 0.05$) between best results and baselines

Mechanism	Grammar and Fluency	Context Relevance	Logic Consistency	Quality
4-team CTC	**2.967**	**2.850**	**2.908**	**2.908**
- Hierarchy Partitioning	1.906	2.219	2.688	2.271
- Feature Extraction	2.096	2.183	2.621	2.300
8-team CTC	**3.000**	**3.250**	**3.000**	**3.083**
- Hierarchy Partitioning	2.255	2.354	2.758	2.456
- Feature Extraction	2.115	2.256	2.653	2.341

Table 5: Ablation study on 4 Teams CTC and 8 Teams CTC. The '+' symbol represents the adding operation and - denotes the removing operation. The highest scores are highlighted in **bold**, and the second-highest scores are presented with underline in each sub-table.

Content Feature Extraction on story generation. Our findings, as detailed in Table 5, reveal significant decrements in story quality. The absence of these mechanisms led to a notable struggle for agents to assimilate diverse team stories within a single aggregation. Without a role-assigned agent for feature extraction, it weakened the overall optimization of story quality. These results demonstrate the indispensable nature and generalization capability of these mechanisms across different domains.

5.5 Case Study

Figures 4 and 5 present the graphical user interfaces (GUIs) of a Tetris game generated by CTC and ChatDev. The Tetris game created using the CTC framework successfully generated a playable game on its first attempt. In this version, the blocks can be moved, rotated, and accelerated. Blocks that reach the bottom of the game matrix change from red to blue and are eliminated if the bottom row is filled, indicating a successful game mechanic. In contrast, the game generated by ChatDev struggled to produce a functional program. Even after a trial-and-error process, the resulting program had significant limitations. Specifically, the square block remained stationary in its initial position and was unable to perform any operations, highlighting the challenges of the ChatDev paradigm in generating a usable program.

6 Limitations

Our study has explored the cooperative behaviors of multiple autonomous agent teams in software development and story generation, yet both researchers and practitioners must be mindful of certain limitations and risks when using the approach to develop new techniques or applications.

Firstly, the framework's dependence on a greedy pruning mechanism could inadvertently lead to the discarding of potentially valuable insights. This is due to the imperfections inherent in evaluation metrics. While the mechanism aims to eliminate low-quality content, it may also prematurely exclude creative solutions that could evolve into high-quality outcomes with further development. There is a trade-off between the efficiency of the pruning process and the potential loss of innovative ideas, which suggests the need for more effective automated evaluation methods in the future, not limited to the domains of software development and story generation.

Secondly, when evaluating the capabilities of autonomous agents from a software development standpoint, it is prudent to avoid overestimating their software production abilities. Our observations indicate that while Cross-Team Collaboration (CTC) significantly improves the quality of both software development and story generation

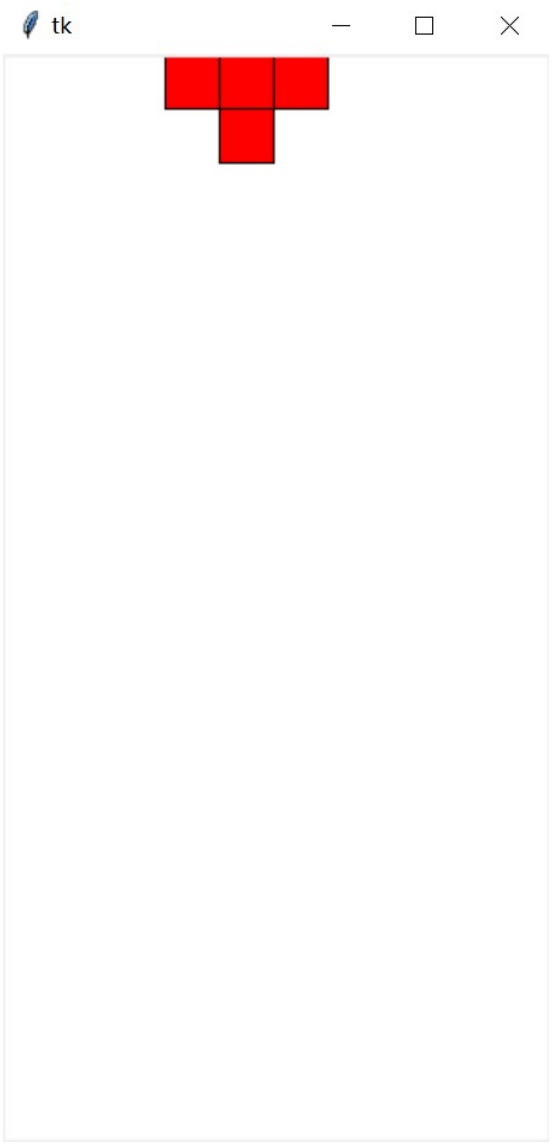

Figure 4: The screenshot of the software generated by ChatDev.

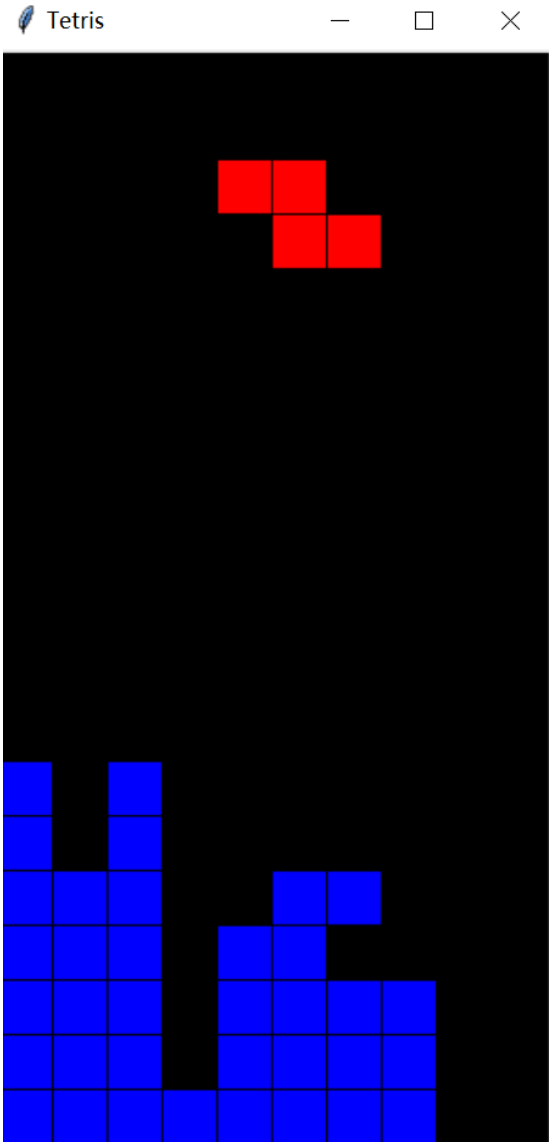

Figure 5: The screenshot of the software generated by our method.

tasks, autonomous agents often default to implementing the most straightforward logic during the software creation process. In the absence of explicit and clear requirements, agents struggle to autonomously discern the underlying concepts and nuances of the task requirements. For example, when developing a Flappy Bird game, if the task guidelines are not meticulously defined, agents may default to representing the bird and tubes with a rudimentary rectangular shape. Similarly, in the construction of an information management system, agents may opt to hard-code the information to be queried in a basic key-value format directly into the code, rather than employing a more sophisticated and flexible external database solution. Therefore, we advocate for the precise definition of detailed software requirements. This includes specifying whether a user interface is essential, if there is a need for the automatic generation of game character assets, or if an external database is necessary. Given the current capabilities of autonomous agents, fulfilling highly detailed requirements is not always assured, underscoring the importance of striking a balance between specificity and practical feasibility in the requirements. In the field of story generation, due to its literary nature, complex task relationships, scene descriptions, and background settings are often required. However, providing agents with overly complex requirements can lead to suboptimal narrative outcomes, as agents may find it challenging to effectively manage and prioritize the various narrative elements during the

writing process. In conclusion, the research on autonomous agents for software and story generation is still in its early stages, and the associated technologies are not yet readily adaptable to complex real-world scenarios. As a result, the current application of these technologies is more suited to the development of prototype systems rather than fully-fledged, real-world software and narrative systems.

Thirdly, the complexity of coordinating multiple teams and managing the communication load increases with the number of teams involved. As the framework scales, the computational and logistical demands rise, which may impact the practicality of applying our framework to very large-scale problems or in resource-constrained environments. Future work is needed to optimize the scalability of the framework while maintaining its efficacy.

7 Conclusion

Recognizing the inherent limitation of a single-team in obtaining and leveraging insights from other teams when completing complex tasks like software development, we introduce a novel multi-team framework termed Cross-Team Collaboration. This framework carefully orchestrates multiple teams with the same software requirement that allows different teams to jointly propose diverse task-oriented decisions, communicate at key phases, and collaboratively aggregate into a final superior software. Our quantitative analysis has effectively demonstrated significant improvements in the software quality. We anticipate that our insights will initiate a paradigm shift in shaping the design of LLM agents into multi-team, propelling agents towards achieving greater software generation quality, and extend it in a broader range of complex tasks, including both programming language generation and natural language generation.

Here, we list several main findings as follows. 1) Cross-team communication for insights interchange significantly improves software quality, indicating the effectiveness of multi-team task handling. It mainly contributes to an appropriate increase in the diversity and effective grouping of content. 2) As the number of participating teams increases, the quality of software is subject to diminishing returns and may even deteriorate. In our study, this is primarily attributed to the increased probability of low-quality software with more teams, which adversely affects the aggregated software quality. The pruning mechanism we introduced effectively addresses this issue. 3) Our CTC framework has the potential for development in broader content generation domains, including natural language generation and programming language generation.

Future research will delve into exploring a broader range of configurations, including both greedy and non-greedy methods for partition. Additionally, we will aim to refine and optimize our evaluation metrics to ensure more precise assessments of automatic software development. In terms of inter-group communication, we plan to introduce and evaluate a variety of communication paradigms, such as Debate, to foster richer and more dynamic interactions. Furthermore, we aspire to apply our approach to enhance a wider array of content generation tasks across various real-world applications, thereby demonstrating the versatility and efficacy of our methods in practical settings.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Mansi Agnihotri and Anuradha Chug. 2020. A systematic literature survey of software metrics, code smells and refactoring techniques. *Journal of Information Processing Systems*, 16(4):915–934.

Mohammad Alahmadi, Abdulkarim Khormi, Biswas Parajuli, Jonathan Hassel, Sonia Haiduc, and Piyush Kumar. 2020. Code localization in programming screencasts. *Empirical Software Engineering*, 25:1536–1572.

Razvan Azamfirei, Sapna R Kudchadkar, and James Fackler. 2023. Large language models and the perils of their hallucinations. *Critical Care*, 27(1):120.

Rajiv D Banker, Gordon B Davis, and Sandra A Slaughter. 1998. Software development practices, software complexity, and software maintenance performance: A field study. *Management science*, 44(4):433–450.

Victor R Basili. 1989. Software development: A paradigm for the future. In *[1989] Proceedings of the Thirteenth Annual International Computer Software & Applications Conference*, pages 471–485. IEEE.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690.

Thorsten Brants, Ashok Popat, Peng Xu, Franz Josef Och, and Jeffrey Dean. 2007. Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 858–867.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2023. Large language models as tool makers. *arXiv preprint arXiv:2305.17126*.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2023. Chateval: Towards better llm-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201*.

Dake Chen, Hanbin Wang, Yunhao Huo, Yuzhao Li, and Haoyang Zhang. 2023a. Gamegpt: Multi-agent collaborative framework for game development. *arXiv preprint arXiv:2310.08067*.

Jiaao Chen, Jianshu Chen, and Zhou Yu. 2019. Incorporating structured commonsense knowledge in story completion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6244–6251.

Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei Zaharia, and James Zou. 2024. Are more llm calls all you need? towards scaling laws of compound inference systems. *arXiv preprint arXiv: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. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, et al. 2023b. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. *arXiv preprint arXiv:2308.10848*.

Cyril Chhun, Fabian M Suchanek, and Chloé Clavel. 2024. Do language models enjoy their own stories? prompting large language models for automatic story evaluation. *arXiv preprint arXiv:2405.13769*.

Roi Cohen, May Hamri, Mor Geva, and Amir Globerson. 2023. Lm vs lm: Detecting factual errors via cross examination. *arXiv preprint arXiv:2305.13281*.

Shiying Ding, Xinyi Chen, Yan Fang, Wenrui Liu, Yiwu Qiu, and Chunlei Chai. 2023. Designgpt: Multi-agent collaboration in design. *arXiv preprint arXiv:2311.11591*.

Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. 2023. Self-collaboration code generation via chatgpt. *arXiv preprint arXiv:2304.07590*.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*.

Jinhao Duan, Renming Zhang, James Diffenderfer, Bhavya Kailkhura, Lichao Sun, Elias Stengel-Eskin, Mohit Bansal, Tianlong Chen, and Kaidi Xu. 2024. Gtbench: Uncovering the strategic reasoning limitations of llms via game-theoretic evaluations. *arXiv preprint arXiv:2402.12348*.

Michael D Ernst. 2017. Natural language is a programming language: Applying natural language processing to software development. In *2nd Summit on Advances in Programming Languages (SNAPL 2017)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik.

Saad Ezzini, Sallam Abualhaija, Chetan Arora, and Mehrdad Sabetzadeh. 2022. Automated handling of anaphoric ambiguity in requirements: a multi-solution study. In *Proceedings of the 44th International Conference on Software Engineering*, pages 187–199.

Ernst Fehr and Simon Gächter. 2000. Cooperation and punishment in public goods experiments. *American Economic Review*, 90(4):980–994.

Ran Gong, Qiuyuan Huang, Xiaojian Ma, Hoi Vo, Zane Durante, Yusuke Noda, Zilong Zheng, Song-Chun Zhu, Demetri Terzopoulos, Li Fei-Fei, et al. 2023. Mindagent: Emergent gaming interaction. *arXiv preprint arXiv:2309.09971*.

Garrett Hardin. 1968. The tragedy of the commons: the population problem has no technical solution; it requires a fundamental extension in morality. *science*, 162(3859):1243–1248.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*.

Shengchao Hu, Li Shen, Ya Zhang, and Dacheng Tao. 2023. Learning multi-agent communication from graph modeling perspective. In *The Twelfth International Conference on Learning Representations*.

Wenyue Hua, Lizhou Fan, Lingyao Li, Kai Mei, Jianchao Ji, Yingqiang Ge, Libby Hemphill, and Yongfeng Zhang. 2023. War and peace (waragent): Large language model-based multi-agent simulation of world wars. *arXiv preprint arXiv:2311.17227*.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2023. Benchmarking large language models as ai research agents. *arXiv preprint arXiv:2310.03302*.

Katja Hutter, Julia Hautz, Johann Füller, Julia Mueller, and Kurt Matzler. 2011. Communitition: The tension between competition and collaboration in community-based design contests. *Creativity and innovation management*, 20(1):3–21.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. 2023. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. *arXiv preprint arXiv:2306.02561*.

Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. 2023. Challenges and applications of large language models. *arXiv preprint arXiv:2307.10169*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023a. Camel: Communicative agents for" mind" exploration of large scale language model society. *arXiv preprint arXiv:2303.17760*.

Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. 2024. More agents is all you need. *arXiv preprint arXiv:2402.05120*.

Yuan Li, Yixuan Zhang, and Lichao Sun. 2023b. Metaagents: Simulating interactions of human behaviors for llm-based task-oriented coordination via collaborative generative agents. *arXiv preprint arXiv:2310.06500*.

Zhongyang Li, Xiao Ding, and Ting Liu. 2018. Generating reasonable and diversified story ending using sequence to sequence model with adversarial training. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1033–1043.

Weixin Liang, Yuhui Zhang, Hancheng Cao, Binglu Wang, Daisy Ding, Xinyu Yang, Kailas Vodrahalli, Siyu He, Daniel Smith, Yian Yin, et al. 2023. Can large language models provide useful feedback on research papers? a large-scale empirical analysis. *arXiv preprint arXiv:2310.01783*.

Jonathan Light, Min Cai, Sheng Shen, and Ziniu Hu. 2023. Avalonbench: Evaluating llms playing the game of avalon. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

Zhiwei Liu, Weiran Yao, Jianguo Zhang, Le Xue, Shelby Heinecke, Rithesh Murthy, Yihao Feng, Zeyuan Chen, Juan Carlos Niebles, Devansh Arpit, et al. 2023. Bolaa: Benchmarking and orchestrating llm-augmented autonomous agents. *arXiv preprint arXiv:2308.05960*.

Cuauhtémoc López-Martín and Alain Abran. 2015. Neural networks for predicting the duration of new software projects. *Journal of Systems and Software*, 101:127–135.

Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. 2022. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. *arXiv preprint arXiv:2209.14610*.

Kaixin Ma, Hongming Zhang, Hongwei Wang, Xiaoman Pan, and Dong Yu. 2023. Laser: Llm agent with state-space exploration for web navigation. *arXiv preprint arXiv:2309.08172*.

Ben Mann, N Ryder, M Subbiah, J Kaplan, P Dhariwal, A Neelakantan, P Shyam, G Sastry, A Askell, S Agarwal, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Nadia Nahar, Shurui Zhou, Grace Lewis, and Christian Kästner. 2022. Collaboration challenges in building ml-enabled systems: Communication, documentation, engineering, and process. In *Proceedings of the 44th international conference on software engineering*, pages 413–425.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*.

Anton Osika. 2023. Gpt-engineer. In *https://github.com/AntonOsika/gpt-engineer*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22.

Giorgio Piatti, Zhijing Jin, Max Kleiman-Weiner, Bernhard Schölkopf, Mrinmaya Sachan, and Rada Mihalcea. 2024. Cooperate or collapse: Emergence of sustainability behaviors in a society of llm agents. *arXiv preprint arXiv:2404.16698*.

Florian Pudlitz, Florian Brokhausen, and Andreas Vogelsang. 2019. Extraction of system states from natural language requirements. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 211–222. IEEE.

Chen Qian, Yufan Dang, Jiahao Li, Wei Liu, Zihao Xie, Yifei Wang, Weize Chen, Xin Cong, Xiaoyin Che, Zhiyuan Liu, and Maosong Sun. 2024a. Experiential co-learning of software-developing agents. In *The 62nd Annual Meeting of the Association for Computational Linguistics*.

Chen Qian, Jiahao Li, Yufan Dang, Wei Liu, YiFei Wang, Zihao Xie, Weize Chen, Cheng Yang, Yingli Zhang, Zhiyuan Liu, et al. 2024b. Iterative experience refinement of software-developing agents. *arXiv preprint arXiv: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. 2024c. Communicative agents for software development. In *The 62nd Annual Meeting of the Association for Computational Linguistics*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023a. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, Xuanhui Wang, et al. 2023b. Large language models are effective text rankers with pairwise ranking prompting. *arXiv preprint arXiv:2306.17563*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

David G Rand and Martin A Nowak. 2013. Human cooperation. *Trends in cognitive sciences*, 17(8):413–425.

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

Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi, Hangyu Mao, Xingyu Zeng, and Rui Zhao. 2023. Tptu: Task planning and tool usage of large language model-based ai agents. *arXiv preprint arXiv:2308.03427*.

Steve Sawyer and Patricia J. Guinan. 1998. Software development: Processes and performance. *IBM systems journal*, 37(4):552–569.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

Murray Shanahan, Kyle McDonell, and Laria Reynolds. 2023. Role play with large language models. *Nature*, 623(7987):493–498.

Theodore R Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L Griffiths. 2023. Cognitive architectures for language agents. *arXiv preprint arXiv:2309.02427*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Chengcheng Wan, Shicheng Liu, Sophie Xie, Yifan Liu, Henry Hoffmann, Michael Maire, and Shan Lu. 2022. Automated testing of software that uses machine learning apis. In *Proceedings of the 44th International Conference on Software Engineering*, pages 212–224.

Yao Wan, Zhou Zhao, Min Yang, Guandong Xu, Haochao Ying, Jian Wu, and Philip S Yu. 2018. Improving automatic source code summarization via deep reinforcement learning. In *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*, pages 397–407.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Lei Wang, Jingsen Zhang, Xu Chen, Yankai Lin, Ruihua Song, Wayne Xin Zhao, and Ji-Rong Wen. 2023b. Recagent: A novel simulation paradigm for recommender systems. *arXiv preprint arXiv:2306.02552*.

Qineng Wang, Zihao Wang, Ying Su, Hanghang Tong, and Yangqiu Song. 2024. Rethinking the bounds of llm reasoning: Are multi-agent discussions the key? *arXiv preprint arXiv:2402.18272*.

Shenzhi Wang, Chang Liu, Zilong Zheng, Siyuan Qi, Shuo Chen, Qisen Yang, Andrew Zhao, Chaofei Wang, Shiji Song, and Gao Huang. 2023c. Avalon's game of thoughts: Battle against deception through recursive contemplation. *arXiv preprint arXiv:2310.01320*.

Song Wang, Nishtha Shrestha, Abarna Kucheri Subburaman, Junjie Wang, Moshi Wei, and Nachiappan Nagappan. 2021. Automatic unit test generation for machine learning libraries: How far are we? In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1548–1560. IEEE.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P Xing, and Zhiting Hu. 2023d. Promptagent: Strategic planning with language models enables expert-level prompt optimization. *arXiv preprint arXiv:2310.16427*.

Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. 2023e. Unleashing cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. *arXiv preprint arXiv:2307.05300*, 1(2):3.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022a. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022b. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Lilian Weng. 2023. Llm-powered autonomous agents.

Anita Williams Woolley, Christopher F Chabris, Alex Pentland, Nada Hashmi, and Thomas W Malone. 2010. Evidence for a collective intelligence factor in the performance of human groups. *science*, 330(6004):686–688.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

Yuzhuang Xu, Shuo Wang, Peng Li, Fuwen Luo, Xiaolong Wang, Weidong Liu, and Yang Liu. 2023. Exploring large language models for communication games: An empirical study on werewolf. *arXiv preprint arXiv:2309.04658*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2023. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2024. Gpt4tools: Teaching large language model to use tools via self-instruction. *Advances in Neural Information Processing Systems*, 36.

Zhangyue Yin, Qiushi Sun, Cheng Chang, Qipeng Guo, Junqi Dai, Xuan-Jing Huang, and Xipeng Qiu. 2023. Exchange-of-thought: Enhancing large language model capabilities through cross-model communication. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15135–15153.

An Zhang, Leheng Sheng, Yuxin Chen, Hao Li, Yang Deng, Xiang Wang, and Tat-Seng Chua. 2023. On generative agents in recommendation. *arXiv preprint arXiv:2310.10108*.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. 2023a. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

Wangchunshu Zhou, Yuchen Eleanor Jiang, Long Li, Jialong Wu, Tiannan Wang, Shi Qiu, Jintian Zhang, Jing Chen, Ruipu Wu, Shuai Wang, et al. 2023b. Agents: An open-source framework for autonomous language agents. *arXiv preprint arXiv:2309.07870*.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, et al. 2023. Ghost in the minecraft: Generally capable agents for open-world enviroments via large language models with text-based knowledge and memory. *arXiv preprint arXiv:2305.17144*.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jurgen Schmidhuber. 2024. Language agents as optimizable graphs. *arXiv preprint arXiv:2402.16823*.