Multi-Agent Collaboration via Cross-Team Orchestration

Zhuoyun Du[†♣♮] **Chen Qian**[†◇] **Wei Liu**[♯] **Zihao Xie**[★] **Yifei Wang**[★] **Rennai Qiu**[★]
Yufan Dang[★] **Weize Chen**[★] **Cheng Yang**[♠✉] **Ye Tian**[♭] **Xuantang Xiong**[♭] **Lei Han**[♭]
[♣]State Key Lab of CAD & CG, Zhejiang University
[♮]Zhejiang Polytechnic Institute, Polytechnic Institute, Zhejiang University
[◇]Shanghai Jiao Tong University [♯]King's College London [★]Tsinghua University
[♠]Beijing University of Posts and Telecommunications [♭]Tencent Robotics X
duzy@zju.edu.cn qianc@sjtu.edu.cn yangcheng@bupt.edu.cn

Abstract

Large Language Models (LLMs) have significantly impacted various domains, especially through organized LLM-driven autonomous agents. A representative scenario is in software development, where agents can collaborate in a team like humans, following predefined phases to complete sub-tasks sequentially. However, for an agent team, each phase 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 leading to suboptimal results or extensive trial and error. To address this, we introduce ***Cross-Team Orchestration*** (***Croto***), a scalable multi-team framework that enables orchestrated teams to jointly propose various task-oriented solutions and interact with their insights in a self-independence while cross-team collaboration environment for superior solutions generation. Experiments reveal a notable increase in software quality compared to state-of-the-art baselines. We further tested our framework on story generation tasks, which demonstrated a promising generalization ability of our framework in other domains. The code and data is available at https://github.com/OpenBMB/ChatDev/tree/macnet

1 Introduction

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), and software development (Richards, 2023; Dong et al., 2024; Zhang et al., 2024a). However, limitations like hallucinations inherent in their standalone capabilities (Richards, 2023), impede LLM's ability to generate usable content for task solving when confronted with complexities surpassing mere chatting. A noteworthy breakthrough lies in the LLM-based collaborative autonomous agents (Park et al., 2023; Li et al., 2023; Wu et al., 2024; Shinn et al., 2024). Typical methods (Qian et al., 2024c; Hong et al., 2023) decompose tasks into several distinct sub-tasks. An instructor gives instructions and an assistant responds with a solution to solve each sub-task. Through a chained multi-turn dialog, they collaboratively generate content (*e.g.,* software, outline, scientific conclusion) for the task. The content produced can vary across multiple iterations given the same task, reflecting the dynamic nature of the problem-solving process by agents (Qian et al., 2024c). A series of autonomous agents interacting through multiple configurable task-oriented phases is the state-of-the-art single-team approach. The team completes the generation process through multiple sequential phases and generates task-oriented data (such as requirement documents and codes), which can be regarded as a decision path.

However, a single team can only execute all phases sequentially according to its predefined configuration (e.g., the number of agents, agent profiles, and LLM hyperparameters), and its decision path is fixed (Qian et al., 2024c; Hong et al., 2023). This design may lead to repetitive errors with a specific configuration when encountering a particular type of problem, hindering self-correction. Furthermore, it limits the agents' ability to explore more diverse and effective decision paths. While graph-like paradigms self-organize agents through dynamic optimization of nodes and edges (Zhuge et al., 2024), they require extensive task-specific customization for all nodes and edges. This complexity complicates their usage and hinders seamless generalization to heterogeneous downstream tasks, making them impractical in many scenarios. Additionally, organizing agents into a graph structure may reduce the task-solving independence of

†Equal Contribution.
✉Corresponding Author.

agents, which is crucial for fostering diverse insights into solutions.

Therefore, it is beneficial to introduce multiple agent teams that are aware of each other, enabling them to collaborate effectively to explore more potential paths without needing specific adjustments while also maintain their independence as a team process self-sufficient. 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 ***Cr****oss-****T****eam* ***O****rchestration* (**Croto**), **a framework that carefully orchestrates different single teams into a multi-team collaborative structure**, each team has the same task assignment to communicate in a collaborative environment. Specifically, our framework enables different teams to independently propose various task-oriented solutions as insights (single-team proposal) and then communicate for insights interchange in some important phases (multi-team aggregation) that boost subsequent task resolution. Different solutions from various teams are divided into groups by a ***Hierarchy Partitioning*** mechanism and then synthesized by a ***Greedy Aggregation*** mechanism that aggregates various solutions and insights into a superior one collaboratively.

Through our experiments with 15 tasks from different categories and styles 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 to bolster 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), 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 Orchestration (Croto), a scalable multi-team collaboration framework that efficiently orchestrates agents into multiple teams to perform cross-team communications, which facilitates seamless content exchange among 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 partitioning and aggregation process of diverse content from multiple teams into a superior outcome, which effectively incorporates multidimensional solutions by retaining their strengths and eliminating their weaknesses.

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

2 Related Work

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; Liu et al., 2024). A notable breakthrough lies in the LLM-based autonomous agents (Wang et al., 2023; Richards, 2023; Osika, 2023), where these agents exhibit proficiency in planning (Chen et al., 2023; Yao et al., 2024), memory (Park et al., 2023; Wang et al., 2024b), and tool use (Qin et al., 2023; Yang et al., 2024a; Qin et al., 2024), thus enabling independent operation within intricate real-world contexts (Zhao et al., 2024; Zhou et al., 2023; Zhang et al., 2023; Weng, 2023), thereby transforming fundamental LLMs into versatile autonomous agents (Shinn et al., 2024; Zhao et al., 2024; Lin et al., 2024; Mei 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; Wang et al., 2024c; Qian et al., 2024d; Zhou et al., 2024), which has widely propelled progress across various domains such as software development (Qian et al., 2024c; Hong et al., 2023), medical treatment (Tang et al., 2023; Li et al., 2024a), educational teaching (Zhang et al., 2024b) and embodied control (Chen et al., 2024).

In contrast to simple majority voting, where agents act independently (Qian et al., 2024b), the concept of collective emergence (Woolley et al., 2010; Hopfield, 1982; Minsky, 1988) suggests that effective collaboration should form an integrated system that fosters interdependent interactions and thoughtful decision-making (Li et al., 2024b; Piatti et al., 2024). Recent research has focused on differentiating agents into distinct roles and encouraging interactions for diverse tasks solving or complex simulation (Xi et al., 2025; Li et al., 2024a; Gao et al., 2024; Yang et al., 2024b; Wang et al., 2025). Studies on exploring organizing agents in hierarchical tree structures for information propagation (Chen et al., 2023; Wang et al., 2024a) or in graph-based structures with predefined nodes and edges (Zhuge et al., 2024) demonstrating that increasing the number and diversity of agents can enhance performance in multi-agent systems. MACNET (Qian et al., 2024d) revealed that the quality of solutions follows a logistic growth pattern as the number of agents scales. Unlike these approaches, our work envisions multi-agent teams as collaborative units, enabling both inter- and intra-team collaborations for optimized solutions generation.

3 Preliminaries

Definition 1 (Chain as a Team) *A single-team ($\mathcal{C}$) is conceptualized as a chain-like structure (Qian et al., 2024c) composed of a series of task-oriented phases ($\mathcal{P}^i$) that sequentially address the resolution of tasks which can be formulated as:*

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

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

Definition 2 (Agent Communication) *In each phase, an instructor agent initiates instructions, instructing ($\rightarrow$) the discourse toward the completion of the subtask, while an assistant agent adheres to these instructions and responds with ($\rightsquigarrow$) appropriate solutions (Li et al., 2023):*

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

Agents engage in a multi-turn dialogue, working collaboratively until they achieve consensus, extracting solutions that can range from the text (e.g., defining a software function point) to code (e.g., creating the initial version of source code), ultimately leading to the completion of the subtask.

Definition 3 (Interaction) *To facilitate intra-team collaboration for task resolution, teams propose their task-oriented solutions from the same phases and jointly participate in solution improvements. We refer to such a collaboration an interaction, which successfully breaks the isolation between teams while preserving their independence.*

4 Methodology

4.1 Cross-Team Orchestration

Facing diverse tasks, a chain-as-a-team often tackles tasks in isolation. While this process is streamlined, it lacks the diversity of insights beneficial to explore a broader range of decision paths for superior task solutions.

A straightforward attempt to break this isolation is to run n teams simultaneously on the same task and ensemble their results. However, this straightforward ensembling approach fails to capitalize on the potential for mutual awareness and collaboration among teams during intermediate phases. This is analogous to exploring n paths in parallel without leveraging the intersections of their intermediate nodes, which could enable the exploration of additional paths. Nevertheless, this approach introduces new challenges, including a significant increase in communication overhead and the risk of incorporating noise from underperforming teams. Moreover, the lack of independence among teams can diminish the diversity of solutions, as teams may converge on similar features.

To alleviate these issues, we propose Croto, a novel cross-team collaborative framework that orchestrates the parallel executions of multiple single-team with configurable temperature and length of chains[1] Each team is assigned the same task objective, and they collectively propose various task-oriented solutions at each phase based on their unique perspectives. At predefined key phases such as design or writing, where critical decisions or significant solution modifications occur, the framework identifies corresponding key phases in other teams. If such phases exist, these teams pause their workflows and extract solutions for cross-team interactions ($\mathcal{E}$). During this interaction process, solutions generated by teams are first grouped and then iteratively aggregated into more refined solutions. This interactive dynamic can be modeled as:

$$\begin{aligned} &\mathcal{N} = \{\mathcal{V}, \mathcal{E}\}, \quad \mathcal{V} = \{v_i \mid i \in \mathcal{I}\} \\ &\mathcal{E} = \{\langle v_i, v_j \rangle \mid \mathbb{I}(v_i) = \mathbb{I}(v_j), v \in \mathcal{K}\} \end{aligned} \tag{3}$$

where $\mathcal{V}$ denotes the set of phases among all teams, indexed by the index set $\mathcal{I}$, $\mathbb{I}(x)$ denotes the name of phase x in a team, and $\mathcal{K}$ denotes the set of key phases. Through Cross-Team Orchestration, a collaborative yet independent cross-team

[1]Length Diversity can be induced manually, meanwhile, can continue to vary autonomously along the process.

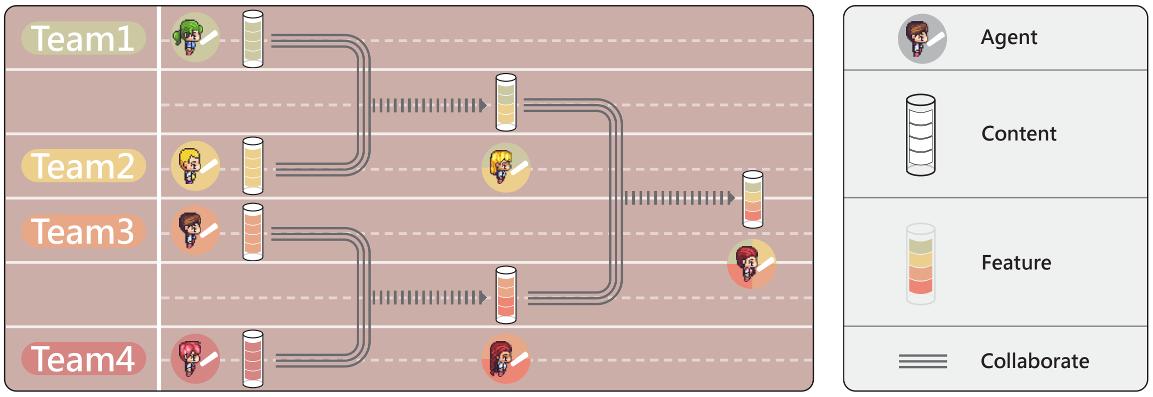

Figure 1: The aggregation process in Cross-Team Orchestration involves multiple agents ( ) from different teams contributing a variety of content ( ). These solutions are partitioned into groups and collaboratively ( ) aggregated through interactions, highlighting the distinctive features ( ) of each team's solution. Ultimately, this process results in a superior outcome that synthesizes the features of all participating teams.

interaction network is established, fostering greater innovation and efficiency in producing superior solutions.

4.1.1 Greedy Aggregation

During interactions, agents collaborate to jointly develop a superior solution. This process is not merely about selecting the best option but focuses on combining the strengths and mitigating deficiencies of all solutions ($\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$), as illustrated in Figure 1. Essentially, it synthesizes multiple decision paths into a single, optimal pathway. To achieve this, we introduce a greedy aggregation mechanism (α) that leverages the features of the solutions. In an aggregation process, a pruning mechanism (θ) filters out a predefined proportion of low-quality solutions[2] to reduce the aggregation burden and enhance the quality of the generated solutions. A role-assigned aggregate agent proficient in synthesizing solutions then meticulously extracts the strengths and weaknesses of each solution. Based on these features, the agent aggregates a superior ($*$) solution that greedily integrates strengths and eliminates weaknesses and explicitly outlines the changes that have been made:

$$s^* = \alpha(\theta(\mathcal{S})) \tag{4}$$

The resulting solution is then disseminated to all teams, replacing prior solutions of each team, guiding subsequent phases of task resolution.

4.1.2 Hierarchy Partitioning

To prevent long-context issues rooted in the overwhelming amount of simultaneous solutions aggregation, meanwhile, enhance the effectiveness of the aggregation process by gradually synthesizing and refining the solutions, we propose Hierarchy Partitioning (τ), as illustrated in Figure 1. This involves grouping solutions from different teams engaged in intra-team interactions and subsequently aggregating them into superior solutions by groups.

Formally, by using uniform partitioning with an expected quantity (u) of solutions per group, a set of collaborative groups ($\mathcal{G}^k = \{g_1^k, g_2^k, \ldots, g^k_{\frac{n}{u^{k+1}}}\}$) are generated. Each group g_i consists of a subset of the solutions from teams that participate in the interaction process:

$$\bigcup_{g \in \mathcal{G}} g_i^k = \mathcal{S}^k, \quad \mathcal{S}^k = \{s_1^k, s_2^k, \ldots, s^k_{\frac{n}{u^k}}\} \tag{5}$$

where k denotes the number of partitioning iterations. Following this, each group of solutions first undergoes an aggregation process, gathered and divided into new groups, and then re-aggregating these aggregated solutions. This iterative process can be formalized as:

$$\begin{aligned} \mathcal{G}^k &= \tau_k(\mathcal{S}^k), \quad \mathcal{S}^{k+1} = \alpha_k(\mathcal{G}^k) \\ s^* &= \alpha_x(\tau_x(\alpha_{x-1}(\ldots \alpha_1(\tau_1(\mathcal{S}^0))))) \end{aligned} \tag{6}$$

This process continues hierarchically, generating aggregated solutions until a single, superior solution remains as the final output.

5 Evaluation

Baselines We chose different types of LLM-driven paradigms as our baselines, which include both single-agent and multi-agent methodologies. **GPT-Engineer** (Osika, 2023) is a foundational

[2]Solutions are evaluated and rated using the Quality metric detailed in Section 5, which effectively enhances our pruning mechanism by eliminating solutions non-arbitrarily.

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**	$0.550^{\dagger}$	$0.779^{\dagger}$	$0.710^{\dagger}$
Croto	👥 👥	$\underline{0.795}$	**0.928**	**0.796**	**0.840**

Table 1: Overall performance comparison of various representative methods, encompassing Single-Agent(🧑), Single-Team Execution (👥), Graph-like (🕸) and Our Cross Team Orchestration (👥 👥) framework. The metrics are the average across all tasks. The highest scores are highlighted in **bold**, and the second-highest scores are presented with $\underline{\text{underline}}$. † indicates significant statistical differences ($p < 0.05$) between baselines and ours.

single-agent method leveraging LLMs for software development, distinguished by its adeptness at swiftly grasping task requirements and applying one-step reasoning to efficiently generate comprehensive solutions, **ChatDev** (Qian et al., 2024c) is an LLM-powered agent collaborative software development framework that organizes the entire software development process into waterfall-style phases, **MetaGPT** (Hong et al., 2023) is an innovative framework that assigns diverse roles to various LLM-powered agents and incorporates standardized operating procedures to facilitate agent collaboration in software development, **AgentVerse** (Chen et al., 2023) is a multi-agent framework that assembles expert agents in structured topologies, using linguistic interactions for autonomous solution refinement, **GPTSwarm** (Zhuge et al., 2024) formalizes a swarm of LLM agents as computational graphs, where nodes represent manually customized functions and edges represent information flow.

Experiment Setup In our experiments, we have validated our framework on heterogeneous tasks, including the scientific domain of software development and the humanities domain of story generation. We employ GPT-3.5-Turbo as the foundational model for its optimal balance of reasoning efficacy and efficiency. We limit communication rounds between agents to a maximum of 5 per phase in each team. By default, the number of teams engaged in the tasks is set to 8 and the temperature parameter is 0.2 with a pruning mechanism. We conduct a pruning mechanism only on 8-team Croto. We configure *coding* and *code completion* phases for software development tasks and after the *writing* phase for story generation tasks as key phases to make cross-team interactions. Our software development experiments randomly draw 15 tasks from the SRDD dataset (Qian et al., 2024c), a collection designed for repository-level software development, and 10 tasks for story generation from ROCStories (Mostafazadeh et al., 2016), a collection of commonsense 5 sentences short stories can be used for longer stories generation. 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 We use four fundamental dimensions to assess specific aspects of the software proposed by previous works (Qian et al., 2024a,c):

- *Completeness* ($\alpha \in [0, 1]$) 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 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 all dimensions above. 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.

5.1 Overall Performance

Table 1 illustrates a detailed comparative analysis of Croto and all baselines. Firstly, the single-team paradigm outperforms the GPT-Engineer, highlighting the benefits of a multi-agent system in decomposing complex task-solving into manageable subtasks. Furthermore, Croto achieved optimal performance with a remarkable improvement over baselines, showing only a slightly lower score in Completeness when compared to the Graph-like paradigm but significantly higher in Executability. The contrast with ChatDev is especially striking, the Completeness score escalates from 0.744 to 0.795, the Executability score witnesses a substantial leap from 0.813 to 0.928, 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 Croto, where the independence of teams maintains their solutions diversity and intra-team collaborations 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

Teams Number Analysis Our investigation on scaling team number, as shown in Figure 2, reveals an intriguing inverse relationship between the Executability and Completeness of the software generated by Croto without the pruning strategy. This finding succinctly captures the essence of the trade-off that is inherent in the framework's performance. Initially, we observed a steady increase in the alignment of codes with task requirements, peaking around the 4-team configuration. This configuration achieves an optimal balance, producing software that is both executable and functionally rich. However, as the number of teams increases beyond four, we notice a gradual decline in Quality.

Mechanism	Completeness	Executability	Consistency	Quality
8-team Croto	0.706†	0.828†	0.792†	0.775†
+ Prune	**0.795**	**0.928**	**0.796**	**0.840**
Δ compared to Vanilla	+0.089	+0.100	+0.004	+0.065
4-team Croto	0.660	0.915†	**0.793**	0.789†
(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 0.4 0.6 0.8)	0.575	0.875†	0.790	0.747†
(0.2 0.2 0.4 0.4)	0.670	**0.925**	0.790	**0.795**
Δ compared to Vanilla	+0.010	+0.010	+0.003	+0.006

Table 2: Investigation of mechanisms in 4-team and 8-team Croto. The temperatures for each team are indicated as (t_1, t_2, t_3, t_4). The '+' symbol represents the adding operation.

Despite this decline, the quality remains superior to that of the single-team configuration, indicating that multi-team collaboration still offers benefits. We hypothesize that the diminishing returns and potential performance decline are due to the agents' difficulty in effectively synthesizing an excessive volume of solution features. To further scale the number of teams without compromising quality, the implementation of a pruning mechanism becomes essential, as it eliminates low-quality solutions before aggregation, effectively reducing the burden on the aggregation agent.

Greedy Pruning To enhance the scalability and performance of Croto, we introduce the Greedy Pruning mechanism, which reduces the aggregation burden and improves the quality of solutions generated by teams. As shown in Table 2, applying pruning in the 8-team configuration achieves the highest scores across all metrics, demonstrating its effectiveness in handling larger team sizes. By evaluating solutions before aggregation, Greedy Pruning eliminates low-quality solutions that could otherwise degrade the final output by introducing suboptimal features and increasing the aggregation burden [3]. This makes our framework more scalable and effective for software development tasks that support Croto in exploring more valuable pathways in a cost-efficient manner while reducing the likelihood of being misled by failed paths.

Temperature Analysis A core idea of our framework is that diverse solutions from multiple teams provide valuable perspectives that, while individually inconspicuous, can be synthesized to positively contribute to task resolution. To evaluate the impact of solution diversity, we varied temperature configurations to enable teams to generate solutions

[3]Burden in the sense that the agent is instructed to synthesize features from all solutions in a group into one solution; more solutions increase the difficulty of aggregation.

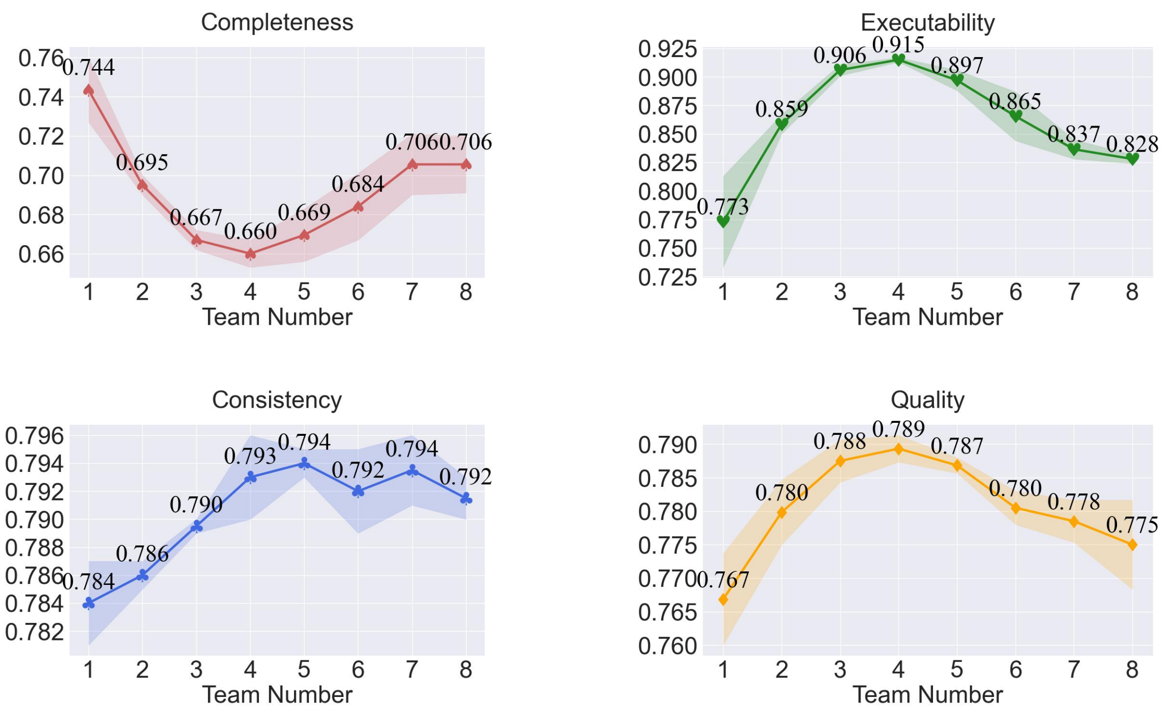

Figure 2: Visualization of result trends concerning team number variations in our framework without greedy pruning mechanism. An upward trend in consistency is observed, along with an inverse relationship between executability and completeness. The highest quality of content is achieved with a team size of four.

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

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

Mechanism	#Token	#Files	#Lines	Duration (s)
Single Team	24377	3.13	104.6	164.36
2-team Croto	32963	3.23	113.4	308.29
3-team Croto	34896	3.85	124.1	532.53
4-team Croto	41903	4.46	135.3	418.34
5-team Croto	44987	4.31	128.5	433.44
6-team Croto	45578	3.95	128.4	461.56
7-team Croto	48812	4.37	126.2	427.08
8-team Croto	52179	4.77	129.6	584.83
Δ compared to single	×2.141	×1.524	×1.239	×3.558

Table 4: Software statistics include Duration (time consumed), #Tokens (number of tokens used), and #Lines (total lines of code per across all files).

with varying degrees of creativity and requirement compliance. As shown in Table 2, an optimal level of diversity significantly improves solution quality. When the temperature is uniform across all teams, the performance gains achieved by Croto are limited. This limitation arises because teams either uniformly prioritize creative but unstable solutions or strictly adhere to rules, resulting in minimal novel insights from cross-team interactions. In contrast, when each team's temperature is set to balance creativity and compliance, Croto demonstrates substantial performance improvements. Analysis of team solutions reveals that cross-team communication often leads to autonomous functional enhancements (e.g., innovative GUI designs, progressively increasing game difficulty), facilitating the integration of beneficial features from diverse solutions.

5.3 Ablation Study

In our ablation study, as presented in Table 3, the removal of Hierarchical Partitioning from the 4-team and 8-team configurations significantly reduced solution quality, from 0.789 and 0.775 to 0.756 and 0.773, respectively. This indicates that, without partitioning into groups, the aggregate agent struggled to effectively handle the diverse features of team solutions in a single aggregation, making it challenging or even impractical to extract and synthesize these features. Furthermore, eliminating role assignment for the aggregate agent further decreased solution quality to 0.735 and 0.744. The absence of structured guidance led to issues such as disorganized solutions, task failures, and feature omissions. These results underscore the importance of our framework's mechanisms in managing complex solutions and ensuring high-quality outputs in multi-team scenarios.

5.4 Statistics Analysis

We present the software statistics in Table 4. Croto generates a greater number of code files and a larger codebase, significantly enhancing the software's functionality and integrity. This trend is consistent across configurations ranging from 2 to 8 teams, demonstrating the effectiveness and scalability of our framework. The increased number of files reflects a more structured programming architecture, resembling software developed by a sophisticated software development team. Although slower and more token-intensive than the single-agent method, our framework remains computationally efficient, as the duration and token consumption do not scale linearly. Specifically, these metrics increase only 2.14 times more tokens and 3.558 times more duration when scaling from a single team to 8 teams. This efficiency is attributed to solution elimination through greedy pruning and the fact that higher-quality aggregated solutions reduce processing complexity in subsequent phases, resulting in fewer average communication rounds per phase and lower token consumption. Considering these factors, we posit that the fundamental characteristics of cross-team software development hold greater significance, outweighing short-term concerns such as time and economic costs in the current landscape.

5.5 Diversity in Collaborative Emergence

The diversity in Croto arises from the interaction among teams, as formalized by the equation:

$$\begin{aligned} p^n(t) &= 1-(1-p(t))^n \\ &\qquad \propto 1-(1-1/r(t))^{|V|^2}, \\ \lim_{|V|\to\infty} p^n(t) &= \lim_{n\to\infty} p^n(t) = 1. \end{aligned} \tag{7}$$

Here, increasing the number of teams ($|V|$) quadratically enhances the likelihood of capturing rare but valuable *long-tail* solution features—such as unconventional yet effective code logic or creative narrative twists—since token distributions in underlying models typically follow a long-tail pattern. This differs from conventional linear agent-level optimization, where rare features are less systematically integrated. The probability $p(t)$ of a rare feature appearing follows a long-tail distribution consistent with Zipf's law (Newman, 2005), such that $p(t) \propto 1/r(t)$, where $r(t)$ denotes the frequency rank of feature t. The sampling size n is proportional to team interaction density ($n \propto |V|^2$), as solution features are aggregated and refined through inter-team comparisons. Consequently, $p^n(t)$, the probability of observing feature t at least once, grows quadratically with $|V|$. As the number of teams increases, the emergence of rare features becomes inevitable, enabling the aggregation process to refine solutions with increasingly nuanced aspects. This mechanism exploits multi-agent interaction scaling to improve solution diversity and quality, aligning with findings in multi-agent debate and cross-examination frameworks (Liang et al., 2023; Du et al., 2023; Cohen et al., 2023).

5.6 Generalizability Analysis

To demonstrate the generalization capability of our framework, we conducted experiments in story generation. The results indicate that our framework significantly enhances the quality of stories generated by both single-agent and single-team execution. This improvement highlights the versatility, robustness, and potential of our framework across diverse domains.

Metrics We evaluate story quality across four critical dimensions:

- ***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*** ($\frac{\omega+\psi+\xi}{3} \in [0, 4]$) is a comprehensive metric that integrates individual dimension scores to provide a comprehensive measure of narrative quality, reflecting the synthesis of language, context, and logic.

Team Number Analysis Experiments on team number shown in Table 5, observe a positive correlation between the number of participating teams and the resultant quality of the generated stories. Notably, the quality demonstrated a substantial

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 Croto	👥👥	2.725	2.800	3.000	2.842
3-team Croto	👥👥	2.967	2.767	2.967	2.900
4-team Croto	👥👥	2.967	2.850	2.908	2.908
5-team Croto	👥👥	2.980	2.880	2.960	2.940
6-team Croto	👥👥	2.983	2.900	2.983	2.956
7-team Croto	👥👥	3.000	3.171	3.014	3.062
8-team Croto	👥👥	3.000†	3.250†	3.000†	3.083†
8-team Croto + Prune	👥👥✂	**3.625**	**3.750**	**3.250**	**3.642**

Table 5: Result trends concerning Team Size Variations in our Framework in Story Generation, encompassing single-agent(🧑), Single-Team Execution (👥) and Cross-Team Orchestration (👥👥) with and without pruning mechanism (✂). † indicates significant statistical differences ($p < 0.05$) between best results and baselines

improvement over outcomes from single agent and single-team baselines, with scores rising from 2.193 and 2.358 to 3.083. However, as the number of teams increased, diminishing returns began to set in. To counteract this trend, we bring in the Greedy Pruning mechanism. This intervention led to a notable enhancement in 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 Croto framework in story generation, suggesting that it is not only beneficial for software development but also generalizes well to creative humanities domains such as narrative generation.

Ablation Study Our ablation study, as illustrated in Table 3, reveals results that align with patterns observed in software development. The removal of partitioning from the 4-team and 8-team configurations resulted in a decline in quality scores, from 2.908 and 3.083 to 2.271 and 2.456, respectively. Similarly, eliminating role assignments for agents further reduced the quality scores from 2.908 and 3.083 to 2.300 and 2.341. While story generation exhibits more literary characteristics compared to software development tasks—which demand precision and error-free execution—they possess a higher tolerance for ambiguity. Unlike software features (*e.g.,* GUI, object-oriented programming), stories encompass more implicit features (*e.g.,* narrative style and thematic intent). These features necessitate that aggregation agents, equipped with assigned roles, process a manageable volume of stories to effectively extract and harmonize these features, thereby producing higher-quality narratives. Given these similarities between the two distinct types of tasks, we hypothesize that content generation tasks (*e.g.,* code, stories, reports, blogs) may similarly benefit from our framework, underscoring its potential broad applicability.

Mechanism	Grammar Fluency	Context Relevance	Logic Consis.	Quality
4-team Croto	**2.967**	**2.850**	**2.908**	**2.908**
- Partition	1,906	2.219	2.688	2.271
- Role	2.096	2.183	2.621	2.300
8-team Croto	**3.000**	**3.250**	**3.000**	**3.083**
- Partition	2.255	2.354	2.758	2.456
- Role	2.115	2.256	2.653	2.341

Table 6: Ablation study on 4 Teams Croto and 8 Teams Croto. The '-' denotes the removing operation. The highest scores are highlighted in **bold**, and the second-highest scores are presented with underline.

6 Conclusion

Recognizing the inherent limitations of a single team in obtaining and leveraging external insights when completing complex tasks such as software development, we introduce a novel multi-team framework called Croto. This framework carefully orchestrates multiple teams with the same task objective, enabling them to jointly propose diverse task-oriented decisions, interact at key phases, and collaboratively aggregate various solutions into a final superior outcome. Without requiring task-specific customization, our quantitative analysis demonstrates significant improvements in outcome quality in software development and story generation, highlighting the framework's scalability and potential generalizability. We anticipate that our insights will initiate a paradigm shift in the design of LLM agents, advancing them toward multi-team collaboration and enhancing solution generation quality across a broader range of complex tasks.

7 Limitations

Our study has explored the collaborative 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 solutions, 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 Orchestration (Croto) significantly improves the quality of both software development and story generation 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 interaction 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.

References

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*.

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.

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

Yongchao Chen, Jacob Arkin, Yang Zhang, Nicholas Roy, and Chuchu Fan. 2024. Scalable multi-robot collaboration with large language models: Centralized or decentralized systems? In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4311–4317. IEEE.

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

Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. 2024. Self-collaboration code generation via chatgpt. *ACM Transactions on Software Engineering and Methodology*, 33(7):1–38.

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*.

Shen Gao, Yuntao Wen, Minghang Zhu, Jianing Wei, Yuhan Cheng, Qunzi Zhang, and Shuo Shang. 2024. Simulating financial market via large language model based agents. *arXiv preprint arXiv:2406.19966*.

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*.

John J Hopfield. 1982. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558.

Akbir Khan, John Hughes, Dan Valentine, Laura Ruis, Kshitij Sachan, Ansh Radhakrishnan, Edward Grefenstette, Samuel R Bowman, Tim Rocktäschel, and Ethan Perez. 2024. Debating with more persuasive llms leads to more truthful answers. *arXiv preprint arXiv:2402.06782*.

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

Junkai Li, Siyu Wang, Meng Zhang, Weitao Li, Yunghwei Lai, Xinhui Kang, Weizhi Ma, and Yang Liu. 2024a. Agent hospital: A simulacrum of hospital with evolvable medical agents. *arXiv preprint arXiv:2405.02957*.

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

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. 2023. Encouraging divergent thinking in large language models through multi-agent debate. *arXiv preprint arXiv:2305.19118*.

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. 2024. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. *Advances in Neural Information Processing Systems*, 36.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Kai Mei, Zelong Li, Shuyuan Xu, Ruosong Ye, Yingqiang Ge, and Yongfeng Zhang. 2024. Llm agent operating system. *arXiv preprint arXiv:2403.16971*.

Marvin Minsky. 1988. *Society of mind*. Simon and Schuster.

Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. 2016. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 839–849.

Mark EJ Newman. 2005. Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351.

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

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*.

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*.

Chen Qian, Zihao Xie, Yifei Wang, Wei Liu, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2024d. Scaling large-language-model-based multi-agent collaboration. *arXiv preprint arXiv:2406.07155*.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, et al. 2024. Tool learning with foundation models. *ACM Computing Surveys*, 57(4):1–40.

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

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Xiangru Tang, Anni Zou, Zhuosheng Zhang, Ziming Li, Yilun Zhao, Xingyao Zhang, Arman Cohan, and Mark Gerstein. 2023. Medagents: Large language models as collaborators for zero-shot medical reasoning. *arXiv preprint arXiv:2311.10537*.

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.

Chenxi Wang, Zongfang Liu, Dequan Yang, and Xiuying Chen. 2025. Decoding echo chambers: Llm-powered simulations revealing polarization in social networks. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 3913–3923.

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

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. 2024a. Mixture-of-agents enhances large language model capabilities. *arXiv preprint arXiv:2406.04692*.

Ruobing Wang, Daren Zha, Shi Yu, Qingfei Zhao, Yuxuan Chen, Yixuan Wang, Shuo Wang, Yukun Yan, Zhenghao Liu, Xu Han, et al. 2024b. Retriever-and-memory: Towards adaptive note-enhanced retrieval-augmented generation. *arXiv preprint arXiv:2410.08821*.

Zhefan Wang, Yuanqing Yu, Wendi Zheng, Weizhi Ma, and Min Zhang. 2024c. Macrec: A multi-agent collaboration framework for recommendation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2760–2764.

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, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. 2024. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling*.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2025. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101.

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

Ziyi Yang, Zaibin Zhang, Zirui Zheng, Yuxian Jiang, Ziyue Gan, Zhiyu Wang, Zijian Ling, Jinsong Chen, Martz Ma, Bowen Dong, et al. 2024b. Oasis: Open agents social interaction simulations on one million agents. *arXiv preprint arXiv:2411.11581*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

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*.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024a. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 1592–1604.

Zheyuan Zhang, Daniel Zhang-Li, Jifan Yu, Linlu Gong, Jinchang Zhou, Zhiyuan Liu, Lei Hou, and Juanzi Li. 2024b. Simulating classroom education with llm-empowered agents. *arXiv preprint arXiv:2406.19226*.

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. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

Wangchunshu Zhou, Yixin Ou, Shengwei Ding, Long Li, Jialong Wu, Tiannan Wang, Jiamin Chen, Shuai Wang, Xiaohua Xu, Ningyu Zhang, et al. 2024. Symbolic learning enables self-evolving agents. *arXiv preprint arXiv:2406.18532*.

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

Appendix

The supplementary information accompanying the main paper provides additional data, explanations and details.

A Algorithm

Here, we provide the pseudocode of our framework for clarity shown in Algorithm 1.

B Code length Study

We term Code length as Complexity (η) which assess the complexity of the generated solutions by quantifying it in terms of the number of lines of generated solutions; A higher number indicates that the completion of the program is more complex.

Delving into the results depicted in Figure 3, we observe that the complexity of the generated programs reaches its zenith at the 4-Team configuration. Subsequently, there is a marginal decline, yet the complexity remains relatively stable. It is evident that under the Team-Weaving paradigm, the complexity of the generated programs consistently surpasses that of the Single-Teaming approach. This observation underscores the efficacy of the Team-Weaving paradigm in enhancing the volume of code generated, thereby opening up possibilities for more extensive software development endeavors.

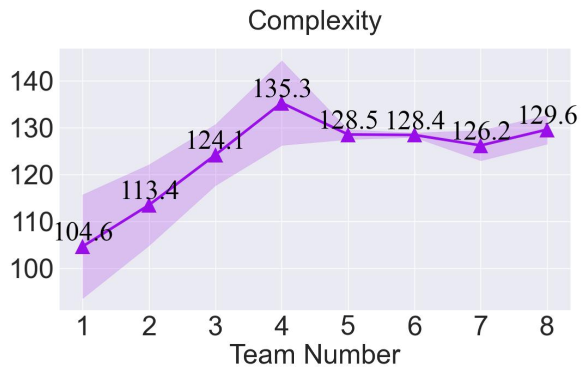

Figure 3: Complexity Trends with Variation in Team Numbers

C Length Diversity of Teams

In contrast to conventional Single Team Execution, where phases within a team are predefined by the user before execution. In Croto, *Waiting Phase* are ($\mathcal{W}$) inserted after key phases where solutions are extracted from teams to perform feature extraction and aggregation lead to a noticeable improvement in solutions quality, resulting in the skipping of certain phases ($\mathcal{P}^i$) in the chat chains ($\mathcal{C}$) of some teams within the Cross-Team Orchestration ($\mathcal{N}$). In Croto, Waiting Phases ($\mathcal{W}$) are inserted after key phases. During these phases, solutions are extracted from teams to perform feature extraction and aggregation, leading to noticeable improvements in solution quality. This process results in the skipping of certain phases ($\mathcal{P}^i$) in the chat chains ($\mathcal{C}$) of some teams within the Cross-Team Orchestration ($\mathcal{N}$). For example, when confronting the code completion phase, the skipping process (S) is invoked when the quality of code from the preceding phase exceeds a predetermined threshold ($\mathcal{Q}^{i-1}$), such as the absence of "TODO" comments. This indicates that the code no longer requires further completion processes, such as function implementation, in subsequent phases, leading to diverse lengths across different teams."

$$\begin{aligned}
&\mathcal{N} = \langle \mathcal{C}^1, \mathcal{C}^2, \dots, \mathcal{C}^{|\mathcal{N}|} \rangle \\
&\mathcal{C}^i = \langle \mathcal{P}^1, \mathcal{P}^2\mathcal{W}^2, \dots, \mathcal{P}^{n-1}\mathcal{W}^{n-1}, \mathcal{P}^n \rangle \\
&\mathsf{S}(\mathcal{P}^i\mathcal{W}^i) \iff \mathcal{P}^{i-1} \geq \mathcal{Q}^{i-1}
\end{aligned} \tag{8}$$

Here, the sequence of phases $\mathcal{P}^i$ interspersed with Waiting Phases $\mathcal{W}^i$ constitutes the chat chain in each team within Cross-Team Orchestration. This characteristic reduces the frequency of LLM calls, lowering development costs and enhancing the framework's efficiency.

D Prompt

D.1 Software Evaluation Prompt

Here, we present the prompts used for evaluating the software quality across multiple teams, as depicted in Figure 4.

D.2 Prompt for Story Quality Evaluation

Here, we present the prompts used for evaluating the story quality across multiple teams, as depicted in Figure 5.

D.3 Generation of Profiles of Programmers

To foster diversity in program generation, it is essential to incorporate a spectrum of programmer profiles. We tasked a Large Language Model (LLM) with the creation of a comprehensive set of 100 features, as depicted in Figures 6, 7, 8, and 9. These features constitute a rich feature pool that programmers may possess.

Software Evaluation Prompt

Prompt

You are an experienced and meticulous software analyst and developer, trusted by others to optimize and enhance their code. Now there are several programs that serve the same purpose and are currently competing in the software development field.
Now you need to carefully analyze and compare these programs to identify their strengths and weaknesses and explain them. Subsequently, you must generate only one new, improved, and runnable program. This program should include the solutions of each file, encompassing the complete code.
Each file must strictly adhere to a markdown code block format. The following tokens must be replaced: "FILENAME" with the lowercase file name, including the file extension; "LANGUAGE" with the programming language; "DOCSTRING" with a string literal specified in the source code for documenting a specific segment of code, and "CODE" is the original code:
FILENAME
“‘LANGUAGE
’’’
DOCSTRING
’’’
CODE
‘‘‘
You will start with the "main" file, then go to the ones that are imported by that file, and so on.
This new program should combine the advantages obtained from the competition between different teams and eliminate all weaknesses.
Please bear in mind that even if some parts of the improved code may similar to the previous program, do not omit them. Instead, clearly write out each line of the improved code. Furthermore, ensure that the code is fully functional, implementing all functions without the use of placeholders (such as ’pass’ in Python).
Here are {Team Number} programs with same task requirement: {Requirements} · · ·

Figure 4: Prompt for Software Quality Evaluation.

Story Evaluation Prompt

Prompt

As a strict StoryMaster, your task is to meticulously evaluate the quality of stories across three primary dimensions: Grammar and Fluency, Context Relevance, and Logic Consistency. Each dimension will be rated on a refined scale from 1 (average) to 4 (perfect), ensuring that only stories of superior quality achieve the highest scores.
Implement Your Evaluation Mechanism with Enhanced Rigor:
Grammar and Fluency (Assess the story's linguistic precision and narrative flow): Score 1 (solid): The story is free of grammatical errors, but the narrative lacks the stylistic variety and eloquence that elevate writing to a higher tier.
Score 2 (proficient): The narrative demonstrates a strong command of grammar and a coherent flow, yet it does not showcase the level of linguistic artistry found in superior works.
Score 3 (excellent): The story exhibits a refined sense of grammar and a compelling narrative flow, with sentence structures that are engaging and demonstrate a high level of craft.
Score 4 (masterful): The story is a testament to linguistic excellence, with sentence structures that are not only clear and elegant but also exhibit a creative and sophisticated use of language that captivates and inspires.
Context Relevance (Examine the coherence, interconnectedness, and depth of solutions within the story):
Score 1 (solid): The story establishes a basic framework of context relevance, but it does not delve into the intricacies of character and thematic development that enrich the narrative.
Score 2 (proficient): The narrative demonstrates a clear connection between elements, yet it lacks the depth and multi-layered solutions that would distinguish it as truly exceptional.
Score 3 (excellent): The story interweaves elements with a high degree of relevance, creating a narrative that is coherent and features solutions that is well-developed and insightful.
Score 4 (masterful): The story achieves an extraordinary level of context relevance, with every element artfully woven into a narrative that is not only coherent but also profound in its exploration of themes and characters, offering a rich and immersive experience.
Logic Consistency (Scrutinize the narrative for logical integrity and internal consistency):
Score 1 (solid): The story maintains a logical structure, but there may be occasional lapses in plausibility or minor inconsistencies that slightly undermine its credibility.
Score 2 (proficient): The narrative is generally logical, with a clear progression of events and character actions, yet it does not reach the level of seamless consistency expected of a superior story.
Score 3 (excellent): The story exhibits a strong logical consistency, with events and character actions that are well-aligned and plausible, contributing to a coherent and believable plot.
Score 4 (masterful): The story is characterized by impeccable logical consistency, with every event and character action meticulously aligned to create a plot that is not only coherent but also demonstrates a deep understanding of causality and human behavior.
After evaluating the story across these dimensions, calculate the overall score by summing the scores from each dimension step by step and present it in the following format: **Score: X** at the end of your evaluation.

Figure 5: Prompt for Story Quality Evaluation.

Algorithm 1 Cross-Team Orchestration

Input: Single-Team Execution set $\mathcal{S}$, key phases set $\mathcal{K}$

Output: Cross-Team Orchestration $\mathcal{N}$, Superior content r_s

```
1:  while $\mathcal{C}^j$ in $\mathcal{S}$ do
2:      for $\mathcal{P}^i_{\mathcal{C}^j}$ in $\mathcal{K}$ do                    ▷ Find key phases in team $\mathcal{C}^j$.
3:          $\mathcal{K} \setminus \{\mathcal{P}^i_{\mathcal{C}^j_k}\}$                          ▷ Remove $\mathcal{P}^i_{\mathcal{C}^j_k}$ from $\mathcal{K}$
4:          $\mathcal{P}^i_{\mathcal{C}^j_k} \to r_j$                                ▷ Content generated by this phase.
5:          $\mathcal{R}^i_x \leftarrow r_j$                                  ▷ Put into communication set, x = 0.
6:          if $|\phi(\mathcal{P}^i_{\mathcal{C}^j_k})| > 1$ then                          ▷ Find it in others.
7:              for team $m$ in $\phi(\mathcal{P}^i_{\mathcal{C}^j_k})$ do
8:                  $\mathcal{E}^i \leftarrow e_m$                            ▷ Connect team $j$ and $m$.
9:                  $\mathcal{P}^i_{\mathcal{C}^m_k} \to r_m \to \mathcal{R}^i_x$
10:             end for
11:             while $|\mathcal{R}^i_x|$ > 1 do
12:                 x = x+1
13:                 $\mathcal{H}(\mathcal{R}^i_x) \to \mathcal{G}$                          ▷ Partition.
14:                 for $g_h$ in $\mathcal{G}$ do
15:                     $g_h \rightarrowtail r^h_s \to \mathcal{R}^i_x$                 ▷ Aggregation
16:                 end for
17:             end while
18:             $r_s = R^i_x[0]$                            ▷ The final unique $r$ in $\mathcal{R}$ is $r_s$.
19:         end if
20:     end for
21: end while
```

Our methodology for profile generation is systematic and probabilistic. Initially, we deterministically select one feature from the first 100. Subsequently, we randomly choose two additional features from the remaining pool, ensuring that the previously selected feature is not repeated. This process yields a unique trio of features for each iteration of profile generation.

When generating a profile, the category of tasks that the programmer is expected to handle is specified to the LLM. The prompt provided to guide the LLM in its generation task is illustrated in Figure 10. This structured approach ensures that each profile is not only distinct but also aligned with the intended task domain, thereby enriching the diversity of the generated programs.

Features A Programmer Could Have

1. Programming Language Proficiency: Mastery of multiple programming languages, such as Python, Java, C++, JavaScript, etc.
2. Algorithm Design and Analysis: Ability to design effective algorithms and analyze their time and space complexity.
3. Data Structure Utilization: Proficiency in using data structures like linked lists, trees, graphs, hash tables, etc., to solve problems.
4. Software Development Methodologies: Familiarity with project management methods such as Agile, Scrum, Kanban, etc.
5. Version Control Operations: Use of tools like Git, SVN, etc., for code version management and team collaboration.
6. Code Testing and Debugging: Writing test cases and conducting unit testing, integration testing, and system testing.
7. Automated Testing Skills: Use of tools like Selenium, JUnit, etc., for automated testing.
8. Database Design and Management: Proficiency in SQL and familiarity with database design, optimization, and management.
9. Front-end Development Skills: Use of technologies like HTML, CSS, JavaScript, etc., for user interface development.
10. Back-end Development Skills: Familiarity with back-end frameworks like Node.js, Django, Spring Boot, etc.
11. Mobile Application Development: Capability in iOS and Android app development, understanding of cross-platform development.
12. Cloud Computing Platform Application: Utilization of cloud services like AWS, Azure, Google Cloud, etc.
13. DevOps Practices: Mastery of CI/CD processes and use of tools like Jenkins, Docker, etc.
14. Containerization Technology: Proficiency in using Docker, Kubernetes for application containerization.
15. Cybersecurity Knowledge: Understanding of cybersecurity fundamentals and implementation of secure coding practices.
16. Artificial Intelligence and Machine Learning: Knowledge and application of AI and machine learning algorithms.
17. Big Data Processing: Familiarity with big data processing technologies like Hadoop, Spark, etc.
18. System Architecture Design: Design of highly available, high-performance, and scalable system architectures.
19. Microservices Architecture Implementation: Building and maintaining microservices-based systems.
20. Project Management: Planning, execution, and monitoring of projects to ensure timely delivery.

Figure 6: Features A Programmer Could Have part 1.

Features A Programmer Could Have

29. Game Development Skills: Development of games using game engines like Unity, Unreal Engine, etc.
30. Blockchain Technology Application: Understanding of blockchain principles and development of blockchain applications.
31. Data Visualization Skills: Use of tools like D3.js, Tableau, etc., for data visualization.
32. Speech and Natural Language Processing: Knowledge of speech recognition and natural language processing technologies.
33. Internet of Things (IoT) Development: Development and integration of IoT devices and applications.
34. Software Quality Assurance: Ensuring software meets quality standards and conducting quality testing.
35. Multithreading and Concurrency Programming: Mastery of multithreading and concurrent programming to improve program efficiency.
36. Distributed System Development: Design and implementation of distributed systems.
37. Software Engineering Principles: Understanding of the fundamental principles and best practices of software engineering.
38. Software Requirements Analysis: Accurate analysis and understanding of software requirements.
39. Application of Design Patterns: Familiarity with and application of common design patterns.
40. Proficiency in Programming Tools: Proficiency in using IDEs, editors, and debugging tools.
41. Static Code Analysis: Use of tools like SonarQube for code quality checks.
42. Software Internationalization and Localization: Development of software supporting multiple languages and cultures.
43. Software Compliance Knowledge: Understanding of software compliance requirements, such as GDPR, HIPAA, etc.
44. Contribution to Open Source Projects: Participation in open source projects and contribution of code and documentation.
45. Software Licensing Management: Understanding of software licensing and copyright laws, and reasonable use of third-party libraries.
46. Software Security Design: Considering security in software development to avoid common vulnerabilities.
47. Software Accessibility: Development of software that meets accessibility standards.
48. Software Maintainability: Writing readable and maintainable code.
49. Software Scalability: Design of scalable software architecture to adapt to future growth.
50. Technical Leadership: Possessing the ability to guide and lead technical teams.
51. Software Testing Strategy: Formulation of effective software testing strategies and plans.
52. Software Configuration Management: Conducting software configuration management to ensure environment consistency.
53. Software Refactoring Techniques: Refactoring existing code to improve code quality.
54. Software Project Management Software: Use of tools like JIRA, Trello, etc., for project management.
55. Software Prototype Design: Creation of software prototypes and conducting user testing and feedback.
56. Software Release Management: Management of software release processes to ensure smooth deployment.

Figure 7: Features A Programmer Could Have part 2.

Features A Programmer Could Have

57. Software Troubleshooting: Quick identification and resolution of software faults.
58. Software Performance Monitoring: Use of monitoring tools to track software performance.
59. Software Security Testing: Conducting security testing to ensure software safety.
60. Software User Training: Providing software usage training and support to users.
61. Software Documentation Management: Management of software documentation to ensure completeness and updates.
62. Software License Compliance: Ensuring compliance of software licenses.
63. Software Version Control Strategy: Formulation and execution of software version control strategies.
64. Software Dependency Management: Management of software dependencies to ensure software stability.
65. Software Build Automation: Automation of the software build process.
66. Software Integration Strategy: Formulation and execution of software integration strategies.
67. Software Testing Automation: Automation of the software testing process.
68. Software Deployment Automation: Automation of the software deployment process.
69. Software Monitoring Automation: Automation of the software monitoring process.
70. Software Maintenance Strategy: Formulation and execution of software maintenance strategies.
71. Software Disaster Recovery Plan: Development of software disaster recovery plans.
72. Software Security Strategy: Formulation and execution of software security strategies.
73. Software Compliance Audit: Conducting software compliance audits.
74. Software Quality Assessment: Assessing software quality to ensure it meets standards.
75. Software Risk Management: Identification and management of software project risks.
76. Software Cost Estimation: Estimation of software development and maintenance costs.
77. Software Resource Planning: Planning of resources required for software development.
78. Software Team Building: Building and managing an efficient software team.
79. Software Communication and Coordination: Effective communication with team members and stakeholders.
80. Software Decision Making: Making wise decisions in the software development process.
81. Software Innovative Thinking: Application of innovative thinking in software development.
82. Software Intellectual Property Protection: Protection of software intellectual property.
83. Software Business Analysis: Analysis of software business requirements and market trends.
84. Software User Experience Design: Design of excellent user experiences.

Figure 8: Features A Programmer Could Have part 3.

Features A Programmer Could Have

85. Software Interaction Design: Design of software interaction processes and interfaces.
86. Software Usability Testing: Conducting software usability testing to enhance user experience.
87. Software Internationalization Strategy: Formulation of software internationalization and localization strategies.
88. Software Compliance Consultation: Providing software compliance consultation services.
89. Software Training and Education: Providing software training for users and team members.
90. Software Technical Support: Providing software technical support and maintenance services.
91. Software Marketing Promotion: Promoting software products and expanding market share.
92. Software Competitive Analysis: Analysis of software market competition and competitors.
93. Software Contract Negotiation: Conducting software contract negotiations and signings.
94. Software Legal Compliance: Ensuring software complies with relevant laws and regulations.
95. Software Ethical Issues: Considering ethical issues in software development.
96. Software Environmental Impact: Assessing the environmental impact of software.
97. Software Social Responsibility: Taking on social responsibility in software development.
98. Software Continuous Improvement: Continuous improvement of software development processes and products.
99. Software Industry Trend Analysis: Keeping up with and analyzing software industry trends
100. Leadership Development in Software: Cultivating and developing leadership in software teams.

Figure 9: Features A Programmer Could Have part 4.

Profile Portray Prompt

: Profile Portray LLM

: Kindly craft a portrayal of a programmer poised to develop a Category Name application, drawing upon the subsequent Number of Features this Programmer Process characteristics. Frame your description in the second person.
Feature 1: <Feature 1>
Feature 2: <Feature 2>
Feature 3: <Feature 3>

Figure 10: Prompt for the Generation of Profiles of Programmers.