Mitigating Collaboration Degeneration in Multi-Agent Code Generation via a Controllable Competitive Collaboration Approach

Shanzhi Gu[1], **Mingyang Geng**[1], **Yihong Dong**[2], **Yunxin Mao**[1], **Zhaoyang Qu**[1], **Hao Zou**[3], **Ruochun Jin**[1], **Zhipeng Liu**[4], **Chuanfu Xu**[1*] and **Haotian Wang**[1*]
[1]National University of Defense Technology
[2]Peking University
[3]Tsinghua University
[4]NorthEastern University
{gushanzhi17, gengmingyang13}@nudt.edu.cn, dongyh@stu.pku.edu.cn,
{maoyunxin, qzy}@nudt.edu.cn, ahio@163.com, jinrc@nudt.edu.cn, 2310543@stu.neu.edu.cn,
{xuchuanfu, wanghaotian13}@nudt.edu.cn

Abstract

Empowered by large language models (LLMs), multi-agent systems (MAS) have shown significant potential in code generation by simulating collaborative workflows. However, we identify a collaboration degeneration phenomenon, where one agent dominates while others remain disengaged, occurring in 38.4% of non-routine HumanEval tasks. Surprisingly, introducing competition among LLM agents eliminates this degeneration and sparks innovation in generated code. Inspired by this catfish effect, we posit that competition helps prevent collaboration degeneration. We thus propose C3, a Controllable Competitive Collaboration framework featuring two novel mechanisms: Centralized Auction-Based Competition (CAB) for role allocation via bidding and Decentralized Communication-Aware Competition (DCC) for solution refinement through opponent-aware adaptation. Evaluated on 70 complex projects from extended SoftwareDev, C3 reduces collaboration degeneration from 32.4% to 12.8%, while enhancing code innovation and diversity. Human evaluations further confirm C3's superiority in functionality, maintainability, and robustness over purely collaborative or competitive baselines.

1 Introduction

Empowered by recent advances in Large Language Models (LLMs), LLM-based agents have become increasingly promising for software development and project management [Dong *et al.*, 2024; Hong *et al.*, 2023; Ishibashi and Nishimura, 2024; Geng *et al.*, 2024; Wang *et al.*, 2023b; Geng *et al.*, 2026], such as Copilot [Perez *et al.*, 2020], Paper2Code [Seo *et al.*, 2025], and Magicoder [Wei *et al.*, 2023]. Notably, generating code for tasks with complex objectives [Bi *et al.*, 2024; Zhang *et al.*, 2024b] or large-scale codebases [Wu *et al.*, 2024; Vergopoulos *et al.*, 2025] increasingly resembles collaborative human workflows. This observation has motivated a growing line of research on structure-aware multi-agent systems (MAS) for code generation [Hong *et al.*, 2023; Islam *et al.*, 2024; Wu *et al.*, 2023].

Existing MAS-based code generation frameworks demonstrate strong potential in emulating human collaboration [Hong *et al.*, 2023; Wu *et al.*, 2023; Islam *et al.*, 2024; Dong *et al.*, 2024; Huang *et al.*, 2023]. For example, MetaGPT [Hong *et al.*, 2023] enforces standardized operating procedures to decompose tasks and assign roles; AutoGen [Wu *et al.*, 2023] coordinates specialized agents through structured dialogues; and MapCoder [Islam *et al.*, 2024] decomposes code synthesis into collaborative stages. We refer to these structurally organized MAS workflows as *Collaborative Code Generation* (Figure 1, left).

However, we observe an overlooked failure mode: existing collaborative frameworks often degenerate into a regime where a single agent dominates iterative interactions while other agents remain largely inactive. In such cases, enforced collaboration effectively collapses into a single-agent multi-round process, which we term **collaboration degeneration**. In a preliminary study on HumanEval [Chen *et al.*, 2021], this phenomenon occurs in 38.4% of non-routine tasks, leading to inefficient or incorrect solutions and undermining the original goal of autonomous MAS collaboration. To understand this issue, we draw inspiration from classical organization theory [Taylor, 1919], which highlights that sustainable cooperation requires a balance between collaboration and competition. Prior studies show that while collaboration provides stability, competition promotes role reallocation, solution diversity, and adaptive reflection, thereby mitigating stagnation [Janis, 1972]. Similar effects have been widely observed in human teams, where controlled competition prevents groupthink and fosters innovation [Riccobono *et al.*, 2016]. These insights motivate a new perspective for MAS-based code generation.

We therefore posit a **pioneer viewpoint**: introducing strategic competition, rather than relying on collaboration alone, is beneficial for MAS-driven code generation. By al-

*Chuanfu Xu and Haotian Wang are the corresponding authors.

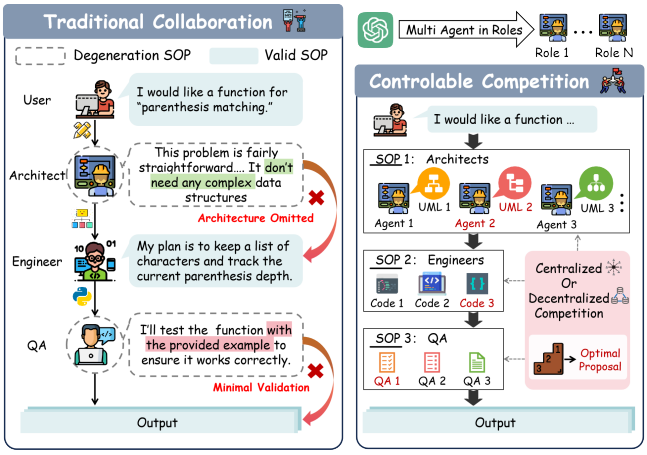

Figure 1: The SOP framework in MetaGPT often leads to collaboration degeneration, where one agent (e.g., Engineer) dominates while others contribute minimally. Our Controllable Competition approach actively mitigates this issue by introducing structured competitive interactions to re-engage all specialized agents.

lowing agents to contest roles or propose alternative strategies when collaboration stalls, competition can break degenerative loops and stimulate diverse solutions. Despite supporting empirical evidence [Mahmud *et al.*, 2025], it remains unclear how to systematically design competition-aware MAS frameworks for code generation. To fill this gap, we propose a *Controllable Competitive Collaboration* (C3) framework, inspired by organization theory and algorithmic game theory, to prevent collaboration degeneration in MAS-based code generation. C3 supports two typical competitive scenarios: (i) centralized competition without inter-agent communication, implemented via a Centralized Auction-Based Competition (CAB) mechanism; and (ii) decentralized competition with communication, realized through a Decentralized Communication-Aware Competition (DCC) mechanism. Our main contributions are summarized as follows:

- We introduce a new perspective that strategic competition complements collaboration in MAS-based code generation, and propose the C3 framework with CAB and DCC mechanisms for centralized and decentralized settings, respectively;
- Extensive experiments on 70 complex tasks show that C3 reduces collaboration degeneration from 32.4% to 12.8%, improves executability by 19.9%, and substantially enhances novelty and diversity, with human evaluations confirming superior functionality, maintainability, and robustness.

2 Related Work

2.1 Single-Agent Empowered Code Generation

Recent advancements in single-agent frameworks have enhanced code generation via architectural and context-aware optimizations [Gu, 2023; Jiang *et al.*, 2024; Li *et al.*, 2024; Zamfirescu-Pereira *et al.*, 2025; Zhang *et al.*, 2024a]. For instance, Self Code Align automates self-alignment through test-driven filtering, boosting performance on HumanEval+ [Wei *et al.*, 2024]. [Gu, 2023] proposes an LLM-based pipeline for compiler fuzzing, improving testcase quality and coverage over traditional methods. PairCoder [Zhang *et al.*, 2024a] introduces a pair-programming-inspired framework with Navigator and Driver agents, achieving strong benchmark results. Beyond these architectural innovations, recent work has also explored leveraging causal inference and state-space models for more robust and individualized predictions, spanning domains from counterfactual estimation to out-of-distribution generalization [Wang *et al.*, 2022; Wang *et al.*, 2023a; Wang *et al.*, 2025].

2.2 Collaborative Multi-Agent Empowered Code Generation

Recent research has explored multi-agent collaboration to enhance code generation through improved reasoning, modularity, and fault isolation [Hong *et al.*, 2023; Dong *et al.*, 2024; Huang *et al.*, 2023; Ishibashi and Nishimura, 2024; Islam *et al.*, 2025; Nunez *et al.*, 2024]. MetaGPT [Hong *et al.*, 2023] employs Standardized Operating Procedures to emulate software engineering workflows. Self-collaboration frameworks organize agents into specialized roles like analysts or testers [Dong *et al.*, 2024]. AgentCoder [Huang *et al.*, 2023] integrates distinct agents for programming and testing, using structured feedback to achieve strong benchmark performance with reduced token cost.

2.3 Competitions in Game Theory: Insight of LLM-Agents

We briefly review two closely related domains from algorithmic game theory: auction theory and non-cooperative competitions. Classical auction mechanisms provide a foundation for incentive design in multi-agent systems [Vickrey, 1961; Myerson, 1981; Milgrom and Weber, 1982; Clarke, 1971], with recent advances integrating deep learning to synthesize revenue-maximizing mechanisms [Dütting *et al.*, 2019; Gemp *et al.*, 2022]. Similarly, non-cooperative game theory offers principles for strategic interactions and equilibrium analysis [Nash, 1950; Nash, 1951; von Neumann and Morgenstern, 1944], with modern multi-agent reinforcement learning frameworks enabling efficient equilibrium approximations [Lowe *et al.*, 2017]. These theoretical foundations suggest that structured competition and cooperation can be mutually reinforcing. Applied to multi-agent LLM code generation, they motivate the design of competitive mechanisms to complement collaboration, fostering innovation and robustness through strategic role interplay.

3 Definition of Collaboration Degeneration

We define collaboration degeneration as a phenomenon where two or more agent's Marginal Utility Contribution (MUC) consistently fall below a predefined threshold τ during multi-agent code generation: if the **marginal utility of two or more agents**, defined as the change in the solution quality due to the agent's input, **remains below the threshold** over several rounds, we classify this as Collaboration Degeneration. We calculate the MUC via the Shapley Value approach (SHAP) [Mosca *et al.*, 2022], since SHAP can isolate

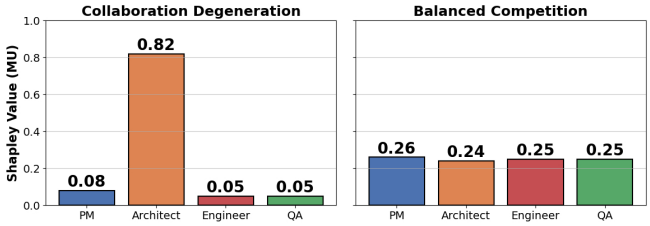

Figure 2: Visualization of the marginal utility contribution.

the intrinsic contribution of each agent from others' contributions. The Shapley Value for agent i in round k is calculated as:

$$\mathrm{MUC}_i^{(k)} = \sum_{S \subseteq \mathcal{R} \setminus \{i\}} \frac{|S|!(|\mathcal{R}| - |S| - 1)!}{|\mathcal{R}|!} [\text{ Quality } (S \cup \{i\}) - \text{ Quality } (S)], \tag{1}$$

where S represents any subset of agents excluding i; Quality (S) is is any subset of agents excluding i; Quality $(S \cup \{i\})$ is the solution quality produced when agent i joins the set S; $|S|$ and $|\mathcal{R}|$ are the sizes of the subsets S and the total set of agents $\mathcal{R}$, respectively. For example, in tasks like algorithmic design, a Product Manager agent may simply repeat the task description without refining functional requirements. A QA Engineer might use only the provided test cases, ignoring edge cases or coverage. The threshold τ is set based on domain-specific quality factors (e.g., correctness), typically set as 0.1. If at least two agents' MUC consistently falls below τ across rounds, collaboration degeneration is treated as occurred.

Collaboration Degeneration in SOP-Driven Frameworks. The MinPath task requires constructing a lexicographically minimal path in a uniquely valued grid. MetaGPT's generic DFS+min, heap solution fundamentally misinterprets the problem: it fails to leverage the injective value domain and incorrectly prioritizes paths, violating lexicographic ordering. This failure stems from collaboration degeneration, a phenomenon observed in 38.4% of non-trivial HumanEval tasks [Chen *et al.*, 2021], where only the Architect agent proposes a naive design while others passively follow, ignoring critical constraints.

Competition as a Catalyst for Improvement. Introducing a basic competitive baseline, where agents of the same role independently propose solutions, demonstrates the potential of competition: the Product Manager surfaces domain constraints, the Architect designs an efficient O(N^2) constructive algorithm, and QA validates with targeted edge-cases. Figure 2 illustrates a shift from skewed to balanced agent contributions, confirming competition mitigates degeneration. However, this unstructured approach only improves 12.1% of all tasks, underscoring the need for more systematic and adaptive competition mechanisms to fully realize its benefits.

4 Methodology

In this section, we design two more refined mechanisms, inspired by centralized and decentralized algorithmic game theory [Shoham and Leyton-Brown, 2008], to instantiate competitions inside the code generation frameworks. We emphasize that all agents in our framework are LLM-based (e.g., GPT-4), autonomously handling code generation through natural language reasoning.

Algorithm 1 Hybrid Competition Mechanisms (CAB & DCC)

Require: Agent set $\mathcal{R}$, task T, metrics M, mode $mode \in \{\text{CAB}, \text{DCC}\}$
Ensure: S^* (CAB) or $\mathcal{S}^*$ (DCC)
if $mode = \text{CAB}$ **then** ▷ Centralized Auction
  for each phase $P \in \{\text{PM}, \text{Arch}, \text{Eng}, \text{QA}\}$ **do**
    for $t = 1$ to r **do**
      $S^* \leftarrow \arg\max_i \text{Score}(S_i^{(t-1)}, M)$ ▷ Coordinator selects best
      $S_i^{(t)} \leftarrow \text{Update}(S_i^{(t-1)}, \text{Feedback}(S^*, S_i^{(t-1)}))$ ▷ Lossers refine
    end for
    Pass S^* to next phase
  end for
  return S^*
else if $mode = \text{DCC}$ **then** ▷ Decentralized Competition
  for each role P in parallel **do**
    for $t = 1$ to r **do**
      for each agent $i \in \mathcal{R}_P$ **do**
        $\mathcal{I} \leftarrow \text{SelectInspiration}(\{S_j^{(t-1)}\}_{j \neq i})$
        $S_i^{(t)} \leftarrow \text{Evolve}(S_i^{(t-1)}, \mathcal{I})$ ▷ Self-adapt using peers
      end for
    end for
    $\mathcal{S}_P^* \leftarrow \{S_i^{(t)}\}$
  end for
  return $\mathcal{S}^* = \bigcup_P \mathcal{S}_P^*$ ▷ Solution pool
end if

4.1 Framework Overview of C3

To address the limitations of purely collaborative workflows, we propose the Controllable Competitive Collaboration (C3) framework, which strategically integrates competition into multi-agent code generation. As illustrated in Figure 3, C3 introduces two competition mechanisms: (1) Centralized Auction-Based Competition (CAB), where a Coordinator Agent orchestrates structured bidding to select optimal proposals (e.g., choosing API designs via weighted scoring); and (2) Decentralized Communication-Aware Competition (DCC), where agents transparently observe and adapt based on peer solutions (e.g., hybridizing architectural designs after reviewing competitors' metrics). A key controllable property lets users steer competition toward specific objectives, such as executability, diversity, or innovation, via pre-defined evaluation metrics [Vergopoulos *et al.*, 2025], enabling targeted and adaptive code generation.

4.2 Centralized Auction-Based Competition (CAB)

The CAB framework balances collaboration and competition through a sequential multi-stage process involving Prod-

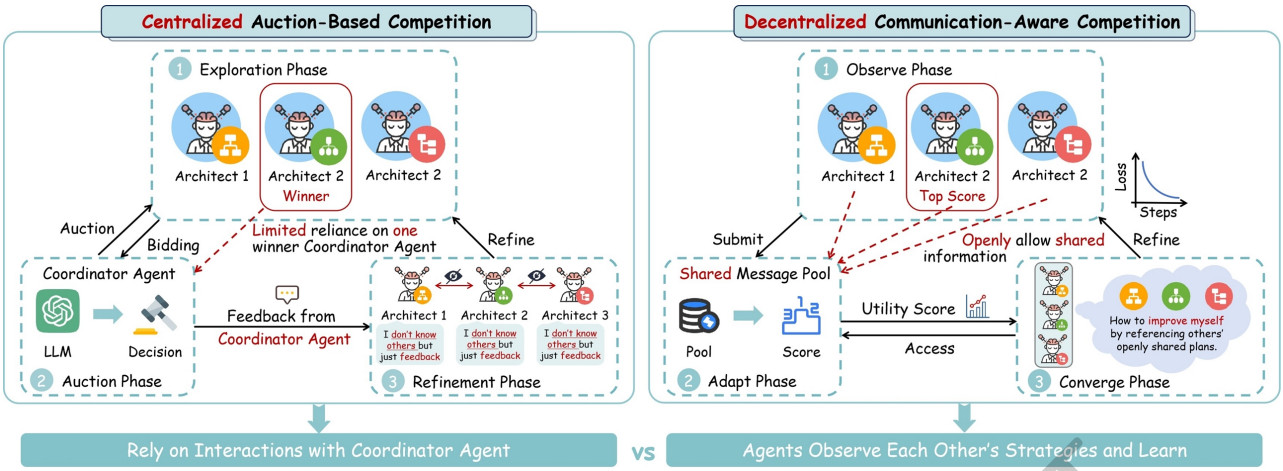

Figure 3: **Top.** The prompts used for evaluating the strategies at different stages. **Left.** The architecture of Centralized Auction-Based Competition, and **Right.** The architecture of Decentralized Communication-Aware Competition.

uct Managers, Architects, Engineers, and QA Engineers. A Coordinator Agent evaluates proposals at each stage using defined metrics, selects the best solution, and provides structured feedback to non-winning agents.

Bidding & Auction. Agents compete within each phase. The Coordinator evaluates proposals based on criteria such as Novelty, Feasibility, and SOP Compliance (PM/Architect phases), Implementability and Performance (Engineer phase), and Test Coverage and Robustness (QA phase). The highest-scoring proposal advances.

Feedback Integration. The process is iterative. After evaluation, the Coordinator provides feedback to losing agents, who refine and resubmit their proposals. This cycle typically runs for 3–5 rounds until agent contributions stabilize, yielding a final solution that is innovative, compliant, and scores highly across key metrics. The overall workflow is illustrated in Figure 3.

Algorithm 1 outlines two hybrid competition modes in the C3 framework. In CAB, a Coordinator agent sequentially selects the best proposal per phase and provides refinement feedback to losers, yielding a single optimized solution S^*. In DCC, agents within each role operate in parallel, observe peer proposals, and self-adapt by incorporating competitors' inspirations, producing a diverse solution pool $\mathcal{S}^*$.

4.3 Decentralized Communication-Aware Competition (DCC)

To systematically understand the design rationale of DCC, we contrast it with the CAB baseline across key architectural dimensions:

Control Structure. CAB employs a centralized Coordinator Agent to govern each phase, while DCC adopts a decentralized, shared-message-pool architecture where agents transparently observe peer outputs and adapt autonomously.

Evaluation Mechanism. Evaluation in CAB is conducted by a single arbiter using predefined metrics. In DCC, evaluation is distributed: each agent scores peer proposals using a standardized rubric, ensuring consistent yet peer-driven assessment.

Feedback & Adaptation. CAB relies on top-down feedback from the coordinator. DCC enables evolutionary adaptation: agents analyze peer proposals, selectively incorporate advantageous elements, and discard weaker strategies through a self-supervised, imitation-based refinement process. This mirrors evolutionary learning, where beneficial traits propagate via competitive peer observation.

Exploration vs. Convergence. DCC balances exploration and convergence through three mechanisms: (1) role-specific standardized evaluation prompts that guide agents toward utility-aligned solutions; (2) empirically grounded limited adaptation rounds (typically 3–5); and (3) mutual visibility of proposals, which promotes convergence on high-scoring strategies via imitation, creating a self-organizing selection pressure.

5 Experiments

5.1 Dataset Contribution: Extended SoftwareDev

Inspired by MetaGPT [Hong *et al.*, 2023], we constructed a benchmark of 70 software development tasks, curated to replicate the categories and proportions described in the original work. Each task, defined by a natural language prompt, covers diverse categories including mini-games, visualization tools, utility applications, algorithmic challenges, and content generators. This dataset provides a comprehensive testbed for evaluating LLM-based agents in end-to-end development workflows, prioritizing diverse agent configurations and task types over isolated algorithmic challenges like HumanEval [Chen *et al.*, 2021]. The task composition is detailed in Table 1.

5.2 Experimental Setting

All methods were evaluated under identical conditions to isolate methodological effects. We used GPT-4o as

Category	Proportion	Example Task	Prompt Summary
Interactive Mini-Games	20/70 (28.6%)	Flappy Bird Game	Implement Flappy Bird in p5.js. The bird flies between pipes and flaps on mouse click. Ends on collision or fall. Score increases with each pipe passed.
Data Viz & Dashboards	15/70 (21.4%)	Weather Dashboard	Python dashboard using OpenWeather API to show current weather and 5-day forecast. Include visualizations for temperature and humidity.
Utility Apps	18/70 (25.7%)	Excel Processor	Streamlit + Pandas app to upload and view Excel files, apply filters, and get summary stats. Should be lightweight with editable output.
Algorithmic Challenges	10/70 (14.3%)	Music Transcriber	Transcribe music audio into sheet music using pitch tracking, neural networks, and chromagram alignment.
Content Generators	7/70 (10.0%)	Press Release Bot	Python tool to fetch updates via APIs/social media and auto-generate press releases with tone/length options. Supports PDF export.

Table 1: Overview of Task Categories in the 70-Task SoftwareDev Benchmark

the base LLM on a homogeneous hardware configuration (8× RTX-4090 GPUs). The benchmark comprised 70 tasks from the SoftwareDev suite, each run with 3 random seeds. For CAB, each phase allowed 2-3 refinement rounds; for DCC, each role performed 3-5 internal adaptation rounds. The evaluation metric set $M = \{\text{Novelty, Feasibility, SOP-Compliance, Performance}\}$ was held constant. A convergence threshold $\epsilon = 0.01$ (relative score change per agent) enabled early termination. Each proposal was constrained to a maximum token length of 4096.

5.3 Research Question

We investigate three research questions to validate the effectiveness of our C3 framework.

- **RQ1: Can C3 mitigate collaboration degeneration in multi-agent code generation?**
- **RQ2: Beyond degeneration prevention, does competition enhance other advantages of generated code artifacts, i.e., novelty and diversity?**
- **RQ3: What are the comparative benefits of centralized versus decentralized competition in multi-agent systems?**
- **RQ4: How can we balance competition and collaboration to maximize overall performance?**
- **RQ5: How do human evaluators perceive the quality of generated software artifacts in real-world case studies?**

5.4 Evaluation Metrics and Baselines

To systematically assess model performance, we first adopt original metrics stated in [Hong *et al.*, 2023]:

- **Executability:** Measures whether generated code runs without syntax or runtime errors, evaluating improvements in degeneration mitigation (**RQ1**).
  Executability $= \frac{\#\text{ of successful executions}}{\#\text{ of total code samples}}$ [Hong *et al.*, 2023], where each code sample is executed in a sandbox with test input; success yields score 1, failure yields 0.
- **Human Revision Cost:** Quantifies the effort required to correct generated code to production quality (**RQ1**, **RQ4**). Revision Cost $= \frac{1}{N}\sum_{i=1}^{N} \text{edit\_score}_i$, where human annotators rate post-editing effort on a 1–10 scale, averaged across samples.

Furthermore, we also employ more extensive dimensions of evaluations to compare and analyze competition and collaboration in below:

- **Degeneration Rate:** The proportion of tasks in which *collaboration degeneration* occurs, defined as cases where the Marginal Utility Contribution of at least two agents consistently falls below the predefined threshold τ (e.g., 0.1), indicating minimal effective contribution from multiple roles.
- **Novelty:** Measures deviation from the reference solution (**RQ2**):
  Novelty $= 1 - \text{AST\_Similarity}(G, R)$, where G is the generated code, R is the reference and similarity is computed by the tree edit distance.
- **Diversity:** Computes the average pairwise AST tree edit distance among 5 executable and minimally correct solutions per task by each method (**RQ2**, **RQ4**):
  Diversity $= \frac{1}{M(M-1)}\sum_{i\neq j} \text{AST\_Dist}(G_i, G_j)$, where pairwise distance between M generated solutions using normalized AST tree edit distance.
- **Task Ownership Entropy (TOE):** Evaluates fairness in contribution distribution (**RQ3**). TOE $= -\sum_{i=1}^{n} p_i \log p_i$, where p_i is the proportion of tasks won by agent i; higher values indicate more balanced participation.
- **Adaptation Responsiveness Rate (ARR):** Captures improvement frequency after peer feedback (**RQ3**). ARR $= \frac{\#\text{ of improved proposals post-feedback}}{\#\text{ of total proposals}}$, where compares proposal quality before and after refinement rounds.

Method	Exec. (%) ↑	Rev. Cost ↓	Deg. Rate (%) ↓
AgentCoder	70.4	7.3	39.7
ChatDev	70.9	6.9	35.1
MetaGPT	72.3	6.4	32.4
NC	78.6	5.2	26.3
CAB	82.1	4.3	18.9
DCC	**86.7** (↑19.9%)	**3.6** (↓47.8%)	**12.8** (↓60.5%)

Table 2: Degeneration Rate vs. Code Quality Comparison, where the relative improvements of DCC are relative to MetaGPT.

- **Feedback Utilization Score (FUS):** Measures how often peer suggestions are reflected in final outputs (**RQ3**, **RQ4**). FUS $= \frac{\#\text{ of peer insights used}}{\#\text{ of insights provided}}$, where counts explicit mentions or implementation of peer strategies in revised proposals.

We choose five representative baseline methods as follows:

- **MetaGPT** [Hong *et al.*, 2023] is a multi-agent framework using SOPs for collaboration, prone to collaboration degeneration.
- **ChatDev** [Qian *et al.*, 2023] is a multi-agent system where LLM agents collaborate via chat across design, coding, and testing.
- **AgentCoder** [Huang *et al.*, 2023] is a multi-agent framework with specialized programmer, test designer, and executor agents that refine code via feedback loops.
- **Naive Competition (NC)** serves as a baseline where agents independently generate and submit solutions; the best is selected based on evaluation metrics. It lacks any collaboration, prioritization, or communication.
- **Isolated Competition (IC)** represents a pure-competition baseline: agents work in complete isolation without communication or shared context. Unlike NC, it excludes all forms of collaboration.

5.5 Experimental Results

RQ1: Effectiveness of C3 in Preventing collaboration degeneration

Table 2 shows that both CAB and DCC significantly reduce collaboration degeneration, with DCC achieving the lowest rate (12.8%) alongside the highest executability (86.7%) and lowest human revision cost (3.6). These results highlight the efficacy of structured competition in improving multi-agent coordination and solution quality. The superior performance of decentralized DCC over centralized CAB—and both over the NC baseline, indicates that fine-grained, peer-aware refinement yields more substantial gains than role-level auctions or unstructured diversity.

RQ2: Gains in Novelty and Diversity

This research question assesses how CAB and DCC enhance solution diversity and novelty. We evaluate across task categories using metrics for novelty, diversity, and semantic distance. As shown in Table 3, competitive mechanisms, especially the decentralized DCC, consistently outperform purely collaborative baselines, yielding more semantically varied and less templated solutions. The advantage is most pronounced in open-ended domains like Creative Content Generators and Algorithmic Challenges, where competition incentivizes exploring non-trivial designs. These results confirm that structured competition not only mitigates collaboration degeneration but actively fosters creativity and innovation.

Category	Method	Novelty (%) ↑	Diversity ↑
Mini-Games	AgentCoder	51.2	0.36
	ChatDev	52.4	0.38
	MetaGPT	54.3	0.42
	NC	57.8	0.49
	CAB	61.4	0.56
	DCC	**66.7** (↑22.8%)	**0.69** (↑64.3%)
Dashboards	AgentCoder	50.7	0.34
	ChatDev	52.0	0.36
	MetaGPT	52.9	0.39
	NC	56.3	0.46
	CAB	60.2	0.52
	DCC	**65.5** (↑23.8%)	**0.65** (↑66.7%)
Utilities	AgentCoder	50.9	0.35
	ChatDev	51.8	0.37
	MetaGPT	53.5	0.41
	NC	56.7	0.47
	CAB	61.0	0.55
	DCC	**66.1** (↑23.6%)	**0.67** (↑63.4%)
Algorithms	AgentCoder	49.8	0.33
	ChatDev	50.6	0.35
	MetaGPT	51.8	0.36
	NC	58.2	0.47
	CAB	63.4	0.54
	DCC	**69.5** (↑34.2%)	**0.70** (↑94.4%)
Creativity	AgentCoder	51.0	0.34
	ChatDev	52.2	0.37
	MetaGPT	55.0	0.40
	NC	60.1	0.50
	CAB	65.7	0.58
	DCC	**71.2** (↑29.5%)	**0.72** (↑80.0%)

Table 3: Diversity and Novelty Comparison by Task Category, where ↑ indicates the relative improvements of DCC over MetaGPT.

Competition Strategy	TOE ↑	ARR ↑	FUS ↑
NC	1.11	0.38	1.7
CAB	1.42	0.61	3.8
DCC	**1.78** (↑25.4%)	**0.75** (↑23.0%)	**5.1** (↑34.2%)

Table 4: CAB vs. DCC Mechanisms, where ↑ means the relative improvements of DCC over CAB.

RQ3: Centralized vs. Decentralized Competition

Table 4 compares NC, CAB, and DCC. DCC outperforms CAB across all metrics, achieving higher Task Ownership Entropy (1.78), Adaptation Responsiveness (0.75), and Feedback Utilization (5.1), indicating richer peer-driven learning. In contrast, CAB offers controlled convergence but limits cross-agent adaptability, while NC performs poorly due to a lack of structured coordination. The results suggest a trade-off: CAB provides structural clarity, whereas DCC fosters emergent collaboration and innovation.

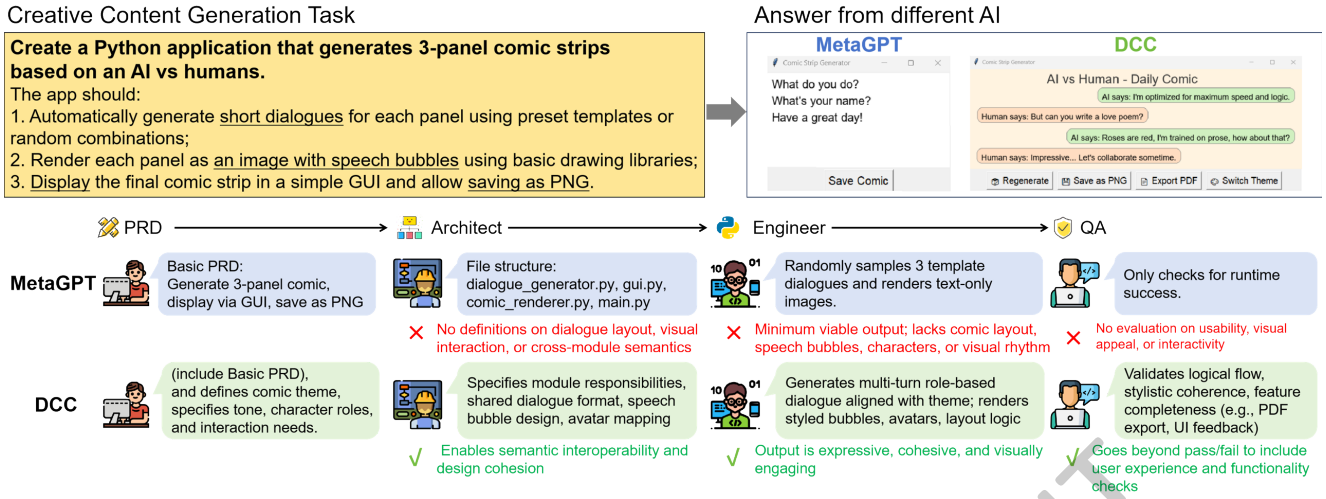

Figure 4: Comparison of MetaGPT and DCC on a comic generation task. The top side shows task requirements and role-level outputs; the bottom side compares final comic outputs from MetaGPT (top) and DCC (bottom), highlighting differences in agent collaboration.

Method	Exec. (%) ↑	Innovation ↑	Diversity ↑	Cost ↓
MetaGPT (Collab-Only)	72.3	53.5	0.41	6.4
IC (Comp-Only)	68.4	**69.5**	**0.73**	7.8
NC (Hybrid)	78.6	57.5	0.48	5.2
CAB (Hybrid)	82.1	61.7	0.55	4.3
DCC (Hybrid)	**86.7**	67.1	0.68	**3.6**

Table 5: Comparison of Collaboration and Competition Strategies.

Method	Functionality ↑	Maintainability ↑	Robustness ↑
MetaGPT	3.2	3.6	2.9
IC	3.4	3.4	2.8
NC	3.5	**3.8**	3.2
CAB	3.8	**3.8**	3.6
DCC	**4.0**	**3.8**	**3.8**

Table 6: Results of Human Evaluation between different collaboration and competition strategies.

RQ4: Balancing Competition and Collaboration

Table 5 examines the collaboration–competition tradeoff. Purely collaborative methods yield lower innovation, while purely competitive ones increase revision costs and reduce executability. Hybrid approaches, especially DCC, best balance executability, innovation, and revision cost, demonstrating the synergy of structured competition built on collaborative foundations.

RQ5: Human Evaluation and Case Study

To complement automatic metrics, we conducted a human evaluation with 10 experienced Python developers on all the 70 cases and evaluated 5 methods, i.e., DCC, CAB, Naive Competition, IC, and MetaGPT, resulting in 350 cases. Each annotator reviewed 35 cases covering all methods, followed by cross-validation to ensure consistency. Ratings were assigned on a 1–5 scale across three dimensions: *functional completeness*, *code readability/maintainability*, and *robustness*. The detailed definitions are: (1) Functional Completeness: Assesses if the solution meets all required features; (2) Code Readability/Maintainability: Evaluates the clarity, organization, and ease of maintaining the code. (3) Robustness: Measures how well the solution handles edge cases, errors.

Human evaluators achieved >94% inter-annotator agreement. As shown in Table 6, DCC excels in functionality (4.0), robustness (3.8), and ties for best maintainability (3.8), demonstrating the strength of structured competition. CAB also performs strongly, particularly in functionality (3.8). In contrast, pure collaboration (MetaGPT) and pure competition (IC) exhibit notably lower robustness (2.9 and 2.8). While IC slightly improves functionality over MetaGPT, its maintainability suffers (3.4) due to lacking architectural coherence. Overall, hybrid competition–collaboration strategies, especially DCC, best balance creativity and stability. A case study from Creative Content Generation (Figure 4) further illustrates this: MetaGPT exhibits collaboration degeneration, producing a basic, unstructured text-only comic from minimal agent coordination.

6 Conclusion

This paper addresses collaboration degeneration in multi-agent code generation, where collaborative systems degrade into single-agent dominance, by introducing the Controllable Competitive Collaboration (C3) framework. C3 integrates Centralized Auction-Based and Decentralized Communication-Aware competition mechanisms to actively foster engagement and innovation. Evaluated on 70 full-stack tasks, C3 significantly improves executability, diversity, and code quality, demonstrating that structured competition catalyzes collaborative intelligence. Future work includes dynamic role switching, cross-domain generalization (e.g., law or science), and human-in-the-loop reinforcement. Our study is the first to formally define and systematically mitigate collaboration degeneration through controlled competitive dynamics.

Ethical Statement

There are no ethical issues.

Acknowledgments

This work is supported in part by the NUDT Youth Independent Innovation Science Fund under Grant No.ZK25-20.

References

[Bi *et al.*, 2024] Zhangqian Bi, Yao Wan, Zheng Wang, Hongyu Zhang, Batu Guan, Fangxin Lu, Zili Zhang, Yulei Sui, Hai Jin, and Xuanhua Shi. Iterative refinement of project-level code context for precise code generation with compiler feedback. *arXiv preprint arXiv:2403.16792*, 2024.

[Chen *et al.*, 2021] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[Clarke, 1971] Edward H. Clarke. Multipart pricing of public goods. *Public Choice*, 11(1):17–33, 1971.

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

[Dütting *et al.*, 2019] Paul Dütting, Zhe Feng, Harikrishna Narasimhan, David C. Parkes, and Sai S. Ravindranath. Optimal auctions through deep learning. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019.

[Gemp *et al.*, 2022] Ian Gemp, Thomas Anthony, Janos Kramár, Tom Eccles, Andrea Tacchetti, and Yoram Bachrach. Designing all-pay auctions using deep learning and multi-agent simulation. *Scientific Reports*, 12:16937, 2022.

[Geng *et al.*, 2024] Mingyang Geng, Shangwen Wang, Dezun Dong, Haotian Wang, Ge Li, Zhi Jin, Xiaoguang Mao, and Xiangke Liao. Large language models are few-shot summarizers: Multi-intent comment generation via in-context learning. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, pages 1–13, 2024.

[Geng *et al.*, 2026] Mingyang Geng, Shanzhi Gu, Zhipeng Liu, Chuanfu Xu, Zhaoyang Qu, and Haotian Wang. Failure localization in multi-agent code generation via knowledge-guided and transferable reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 318–326, 2026.

[Gu, 2023] Qiuhan Gu. Llm-based code generation method for golang compiler testing. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 2201–2203, 2023.

[Hong *et al.*, 2023] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 3(4):6, 2023.

[Huang *et al.*, 2023] Dong Huang, Jie M. Zhang, Michael Luck, Qingwen Bu, Yuhao Qing, and Heming Cui. Agentcoder: Multi-agent code generation with effective testing and self-optimization. *arXiv preprint arXiv:2312.13010*, 2023.

[Ishibashi and Nishimura, 2024] Yoichi Ishibashi and Yoshimasa Nishimura. Self-organized agents: A llm multi-agent framework toward ultra large-scale code generation and optimization. *arXiv preprint arXiv:2404.02183*, 2024.

[Islam *et al.*, 2024] Md Ashraful Islam, Mohammed Eunus Ali, and Md Rizwan Parvez. Mapcoder: Multi-agent code generation for competitive problem solving. *arXiv preprint arXiv:2405.11403*, 2024.

[Islam *et al.*, 2025] Md. Ashraful Islam, Mohammed Eunus Ali, and Md. Rizwan Parvez. Codesim: Multi-agent code generation and problem solving through simulation-driven planning and debugging. In *Findings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2025.

[Janis, 1972] Irving L Janis. *Victims of groupthink: A psychological study of foreign-policy decisions and fiascoes.* Houghton Mifflin, 1972.

[Jiang *et al.*, 2024] Xue Jiang, Yihong Dong, Lecheng Wang, Zheng Fang, Qiwei Shang, Ge Li, Zhi Jin, and Wenpin Jiao. Self-planning code generation with large language models. *ACM Transactions on Software Engineering and Methodology*, 33(7):1–30, 2024.

[Li *et al.*, 2024] Jia Li, Yunfei Zhao, Yongmin Li, Ge Li, and Zhi Jin. Acecoder: An effective prompting technique specialized in code generation. *ACM Transactions on Software Engineering and Methodology*, 33(8):1–26, 2024.

[Lowe *et al.*, 2017] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

[Mahmud *et al.*, 2025] Tarek Mahmud, Bin Duan, Corina Pasareanu, and Guowei Yang. Enhancing llm code generation with ensembles: A similarity-based selection approach. *arXiv preprint arXiv:2503.15838*, 2025.

[Milgrom and Weber, 1982] Paul R. Milgrom and Robert J. Weber. A theory of auctions and competitive bidding. *Econometrica*, 50(5):1089–1122, 1982.

[Mosca *et al.*, 2022] Edoardo Mosca, Ferenc Szigeti, Stella Tragianni, Daniel Gallagher, and Georg Groh. Shap-based explanation methods: a review for nlp interpretability. In *Proceedings of the 29th international conference on computational linguistics*, pages 4593–4603, 2022.

[Myerson, 1981] Roger B. Myerson. Optimal auction design. *Mathematics of Operations Research*, 6(1):58–73, 1981.

[Nash, 1950] John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950.

[Nash, 1951] John F. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951.

[Nunez *et al.*, 2024] Ana Nunez, Nafis Tanveer Islam, Sumit Kumar Jha, and Peyman Najafirad. Autosafecoder: A multi-agent framework for securing llm code generation through static analysis and fuzz testing. *arXiv preprint arXiv:2409.10737*, 2024.

[Perez *et al.*, 2020] Ivan Perez, Frank Dedden, and Alwyn Goodloe. Copilot 3. Technical report, National Aeronautics and Space Administration, 2020.

[Qian *et al.*, 2023] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 2023.

[Riccobono *et al.*, 2016] Francesca Riccobono, Manfredi Bruccoleri, and Andreas Größler. Groupthink and project performance: The influence of personal traits and interpersonal ties. *Production and Operations Management*, 25(4):609–629, 2016.

[Seo *et al.*, 2025] Minju Seo, Jinheon Baek, Seongyun Lee, and Sung Ju Hwang. Paper2code: Automating code generation from scientific papers in machine learning. *arXiv preprint arXiv:2504.17192*, 2025.

[Shoham and Leyton-Brown, 2008] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

[Taylor, 1919] Frederick Winslow Taylor. *The principles of scientific management*. Harper & brothers, 1919.

[Vergopoulos *et al.*, 2025] Konstantinos Vergopoulos, Mark Niklas Müller, and Martin Vechev. Automated benchmark generation for repository-level coding tasks. *arXiv preprint arXiv:2503.07701*, 2025.

[Vickrey, 1961] William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16(1):8–37, 1961.

[von Neumann and Morgenstern, 1944] John von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1st edition, 1944.

[Wang *et al.*, 2022] Haotian Wang, Wenjing Yang, Longqi Yang, Anpeng Wu, Liyang Xu, Jing Ren, Fei Wu, and Kun Kuang. Estimating individualized causal effect with confounded instruments. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1857–1867, 2022.

[Wang *et al.*, 2023a] Haotian Wang, Kun Kuang, Long Lan, Zige Wang, Wanrong Huang, Fei Wu, and Wenjing Yang. Out-of-distribution generalization with causal feature separation. *IEEE Transactions on Knowledge and Data Engineering*, 36(4):1758–1772, 2023.

[Wang *et al.*, 2023b] Shangwen Wang, Mingyang Geng, Bo Lin, Zhensu Sun, Ming Wen, Yepang Liu, Li Li, Tegawendé F Bissyandé, and Xiaoguang Mao. Natural language to code: How far are we? In *Proceedings of the 31st ACM joint European software engineering conference and symposium on the foundations of software engineering*, pages 375–387, 2023.

[Wang *et al.*, 2025] Haotian Wang, Haoxuan Li, Hao Zou, Haoang Chi, Long Lan, Wanrong Huang, and Wenjing Yang. Effective and efficient time-varying counterfactual prediction with state-space models. In *The Thirteenth International Conference on Learning Representations*, 2025.

[Wei *et al.*, 2023] Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with oss-instruct. *arXiv preprint arXiv:2312.02120*, 2023.

[Wei *et al.*, 2024] Yuxiang Wei, Federico Cassano, Jiawei Liu, Yifeng Ding, Naman Jain, Zachary Mueller, Harm de Vries, Leandro Von Werra, Arjun Guha, and Lingming Zhang. Selfcodealign: Self-alignment for code generation. *arXiv preprint arXiv:2410.24198*, 2024.

[Wu *et al.*, 2023] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

[Wu *et al.*, 2024] Qinyun Wu, Chao Peng, Pengfei Gao, Ruida Hu, Haoyu Gan, Bo Jiang, Jinhe Tang, Zhiwen Deng, Zhanming Guan, Cuiyun Gao, et al. Repomastereval: Evaluating code completion via real-world repositories. *arXiv preprint arXiv:2408.03519*, 2024.

[Zamfirescu-Pereira *et al.*, 2025] JD Zamfirescu-Pereira, Eunice Jun, Michael Terry, Qian Yang, and Björn Hartmann. Beyond code generation: Llm-supported exploration of the program design space. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–17, 2025.

[Zhang *et al.*, 2024a] Huan Zhang, Wei Cheng, Yuhan Wu, and Wei Hu. A pair programming framework for code generation via multi-plan exploration and feedback-driven refinement. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pages 1319–1331, 2024.

[Zhang *et al.*, 2024b] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*, 2024.