Experiential Co-Learning of Software-Developing Agents

Chen Qian[†⋆] **Yufan Dang**[†⋆] **Jiahao Li**[♠] **Wei Liu**[⋆] **Zihao Xie**[⋆]
Yifei Wang[⋆] **Weize Chen**[⋆] **Cheng Yang**[♣✉] **Xin Cong**[⋆] **Xiaoyin Che**[♦]
Zhiyuan Liu[⋆✉] **Maosong Sun**[⋆✉]
[⋆]Tsinghua University [♠]Dalian University of Technology
[♣]Beijing University of Posts and Telecommunications [♦]Siemens
qianc62@gmail.com dangyf21@mails.tsinghua.edu.cn
yangcheng@bupt.edu.cn liuzy@tsinghua.edu.cn sms@tsinghua.edu.cn

Abstract

Recent advancements in large language models (LLMs) have brought significant changes to various domains, especially through LLM-driven autonomous agents. A representative scenario is in software development, where LLM agents demonstrate efficient collaboration, task division, and assurance of software quality, markedly reducing the need for manual involvement. However, these agents frequently perform a variety of tasks independently, without benefiting from past experiences, which leads to repeated mistakes and inefficient attempts in multi-step task execution. To this end, we introduce *Experiential Co-Learning*, a novel LLM-agent learning framework in which instructor and assistant agents gather shortcut-oriented experiences from their historical trajectories and use these past experiences for future task execution. The extensive experiments demonstrate that the framework enables agents to tackle unseen software-developing tasks more effectively. We anticipate that our insights will guide LLM agents towards enhanced autonomy and contribute to their evolutionary growth in cooperative learning. The code and data are available at https://github.com/OpenBMB/ChatDev.

1 Introduction

In the ever-evolving field of artificial intelligence, large language models (LLMs) have marked a transformative shift across numerous domains (Vaswani et al., 2017; Brown et al., 2020; Bubeck et al., 2023). Despite their impressive abilities, when dealing with complex situations that extend beyond mere chatting, these models show certain limitations inherent in their standalone capabilities (Richards, 2023). Recent research in *autonomous agents* has significantly advanced LLMs by integrating sophisticated features like context-sensitive memory (Park et al., 2023), multi-step planning (Wei et al., 2022b), and strategic tool use (Schick et al., 2023). This enhancement has expanded their capacity to effectively manage a broader spectrum of complex tasks, including social simulation (Park et al., 2023; Wang et al., 2023f; Hua et al., 2023), software development (Osika, 2023; Qian et al., 2023), game playing (Wang et al., 2023a; Zhu et al., 2023; Wang et al., 2023d; Gong et al., 2023), and scientific research (Huang et al., 2023; Liang et al., 2023). In order to study the cooperative dynamics of autonomous agents more pertinently, we choose software development (Mills, 1976) as an representative scenario, due to its *complexity* that demands a blend of natural and programming language skills (Mills, 1976), the *processuality* that often requires an in-depth understanding of coding and continuous alterations (Barki et al., 1993), and the *objectivity* of code that can provide quantifiable feedback (Compton and Hauck, 2002).

With the development of autonomous-agent technology, a successful breakthrough has been the integration of communication among multiple agents (Park et al., 2023; Li et al., 2023a; Qian et al., 2023). Representative methods epitomize this methodology by segmenting task execution into distinct subtasks. Through engaging in cooperative communication, agents participate in instructive or responsive conversations, collaboratively contributing to the achievement of a cohesive and automated solution for task execution. For example, in ChatDev (Qian et al., 2023), a recent agent-communication framework for software development, a reviewer agent in charge of an external compiler iteratively provides instructions for software optimization (*e.g.,* completing unimplemented code, making functional changes, and debugging programs), to which a programmer agent then reacts to these instructions by appropriately up-

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

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 5628–5640

dating the source code. The development of a more adaptive and proactive approach to task-solving by these agents marks a significant leap in autonomy, going beyond the typical prompt-guided dynamic in human-computer communications (Yang et al., 2024) and substantially reducing dependence on human involvement (Li et al., 2023a; Qian et al., 2023; Wu et al., 2023).

However, when confronted with a diverse range of task types, current multi-agent collaboration methods tend to handle each task independently, which largely stems from the absence of a methodology that can effectively incorporate the experiences accumulated from previously completed tasks (Qian et al., 2023; Chen et al., 2024; Park et al., 2023; Hong et al., 2024). Consequently, the inexperience nature frequently results in repetitive errors or unnecessary trial-and-error processes through multi-step task execution, ultimately necessitating additional human involvement especially when these methods are applied to real-world scenarios.

How do we design, gather, and apply useful experiences to enhance multi-agent collaboration? In this paper, we propose *Experiential Co-Learning*, a novel multi-agent learning paradigm designed to boost agents' software-developing abilities through the utilization of experiences gathered from their historical communications. The method regards agents into two functional roles—instructor and assistant, involving three core modules: 1) the *co-tracking* module promotes communicative rehearsals between the agents, fostering cooperative exploration and the creation of procedural trajectories for various training tasks; 2) the *co-memorizing* module heuristically mines "shortcuts"[1] from historical trajectories using external environment feedback, which are then preserved in their experience pools in an interleaved manner; 3) the *co-reasoning* module encourages agents to enhance their instructions and solutions by utilizing their collective experience pools when facing unseen tasks. The comprehensive assessment of collaborative processes between autonomous agents in diverse software development tasks reveals that the proposed framework significantly boosts collaborative efficiency and reduces the need for extra human involvement.

In summary, our main contributions include:

- To our knowledge, this study is the first to integrate past experiences into the LLM-powered multi-agent collaboration. Through co-tracking, co-memorizing, and co-reasoning, this framework facilitates cooperative learning among two distinct agent types (instructor and assistant) by leveraging heuristical experiences extracted from their historical task execution trajectories.

- We propose to construct task execution graphs based on procedural trajectories, in which "shortcuts" linking non-adjacent nodes are extracted as experiences, which can effectively motivate agents to engage in shortcut thinking during reasoning.

- We conducted extensive experiments from multiple perspectives to validate the effectiveness of our framework. The findings highlight the enhanced quality and efficiency of agents' collaborative behavior in software development.

2 Related Work

Trained on vast datasets to comprehend and manipulate billions of parameters, LLMs have become pivotal in natural language processing (Brown et al., 2020; Bubeck et al., 2023; Vaswani et al., 2017; Radford et al., 2019; Touvron et al., 2023; Wei et al., 2022a; Shanahan et al., 2023; Chen et al., 2021; Brants et al., 2007; Chen et al., 2021; Ouyang et al., 2022; Yang et al., 2024; Qin et al., 2023; Kaplan et al., 2020). Recent progress, particularly in the filed of autonomous agents (Zhou et al., 2024; Wang et al., 2023a; Park et al., 2023; Wang et al., 2023f; Richards, 2023; Osika, 2023; Wang et al., 2024), is largely attributed to the foundational advances in LLMs. These agents utilize the robust capabilities of LLMs, displaying remarkable skills in memory (Park et al., 2023; Sumers et al., 2024), planning (Osika, 2023; Chen et al., 2024; Liu et al., 2023) and tool use (Schick et al., 2023; Cai et al., 2024; Qin et al., 2024; Ruan et al., 2023; Yang et al., 2023), enabling them to operate independently in complex, real-world scenarios (Zhao et al., 2024; Zhou et al., 2024; Ma et al., 2023; Zhang et al., 2023; Wang et al., 2023b; Ding et al., 2023; Weng, 2023). Since agents like Reflexion (Shinn et al., 2023) showcase feedback-based improvement yet often lack cross-task experience, to enable an autonomous agent to improve through trial and error across various training tasks, ExpeL (Zhao et al., 2024) innovatively defines experience as a history

[1]The term "shortcut" is used positively to denote a more efficient pathway, unlike in some papers where it implies a superficial correlation (Geirhos et al., 2020).

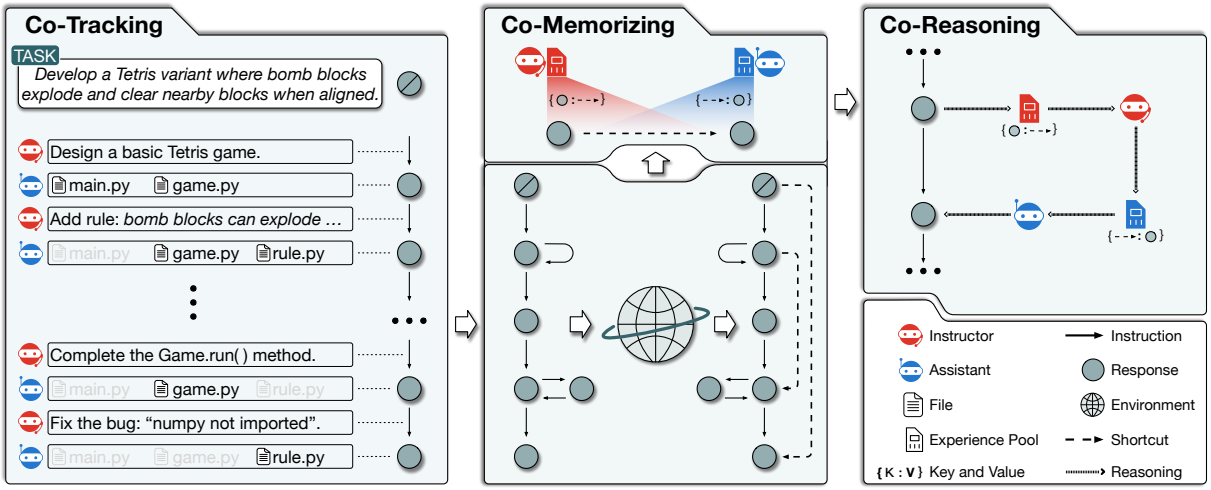

Figure 1: The framework of Experiential Co-Learning. The co-tracking module promotes communicative rehearsals between the agents, fostering cooperative exploration and the creation of procedural trajectories for various training tasks. The co-memorizing module heuristically extracts "shortcuts" from the trajectories under external supervision, integrating these heuristic shortcuts into their collective experience pools. The co-reasoning module combines agents' collective experience pools to foster an communication of augmented instructions and solutions, improving their ability to collaboratively solve unseen tasks.

of successful task trajectories and utilizes these experiences for in-context reasoning.

Parallel to these attempts, the autonomous communication among multiple agents is now a promising paradigm, signaling a shift towards multi-agent collaboration paradigm (Park et al., 2023; Zhou et al., 2023; Chen et al., 2024; Chan et al., 2024; Chen et al., 2023; Cohen et al., 2023; Li et al., 2023b; Hua et al., 2023; Guo et al., 2024). Among them, software-developing agents facilitate the breakdown of complex tasks into more manageable, finer-grained subtasks (Hong et al., 2024; Qian et al., 2023). An instructor issues directional instructions, and an assistant provides relevant solutions, facilitating a streamlined workflow for task execution. This approach not only boosts productivity but also exhibits a degree of quality that surpasses the traditional prompt-guided paradigm in human-computer communications, reducing the need for manual involvement (Li et al., 2023a; Chen et al., 2024).

3 Experiential Co-Learning

Traditional multi-agent collaboration methods often neglect to accumulate experience from past tasks, leading to repetitive errors or unnecessary trial-and-error processes in similar future tasks (Qian et al., 2023). To remedy this, we propose *Experiential Co-Learning*, illustrated in Figure 1, powered by two distinct autonomous agents and comprising three essential modules: 1) the *co-tracking* module establishes a rehearsal collaboration between an instructor and an assistant, focusing on tracking their cooperative "procedural trajectories" for various training tasks, showcasing clear strategies in their communicative collaboration; 2) the *co-memorizing* module heuristically extracts "shortcuts" from the trajectories under external supervision, integrating these heuristic shortcuts into their collective experience pools; 3) the *co-reasoning* module combines the two agents' collective experience pools to foster an communication of augmented instructions and solutions, improving their ability to collaboratively solve unseen tasks.

3.1 Co-Tracking

The co-tracking module sets up a collaborative task execution between an instructor and an assistant, with the goal of tracking procedural trajectories for various training tasks (Qin et al., 2024). Each trajectory captures the dynamic progression of a specific task, detailing the evolving roadmap and clearly illustrating the interacted solutions throughout the task execution process.

Formally, in the set of training tasks $\mathcal{T}$, each task $t \in \mathcal{T}$ fosters a functional interplay of communications between agents, streamlined towards the effective execution of the task. During this procedure, the instructor gives a series of instructions ($\mathcal{I} = \{i_1, i_2, \cdots, i_n\}$), to which the assistant

responds with a matching sequence of solutions ($\mathcal{S} = \{s_1, s_2, \cdots, s_n\}$), where each solution represents an intact software code. This communicative dynamic can be naturally modeled as a directed chain $\mathcal{G} = (\mathcal{N}, \mathcal{E})$:

$$\mathcal{N} = \{s_j | s_j \in \mathcal{S}\} \cup \{s_0\}$$
$$\mathcal{E} = \{(s_j, i_{j+1}, s_{j+1}) | s_j, s_{j+1} \in \mathcal{S}, i_{j+1} \in \mathcal{I}\}$$

where $\mathcal{N}$ represents the nodes corresponding to the solutions (with s_0 denoting the initial, typically empty solution), and $\mathcal{E}$ denotes the edges corresponding to the instructions. Each edge (s_j, i_{j+1}, s_{j+1}) illustrates the transition from one solution s_j to the modified one s_{j+1}, guided by the instruction i_{j+1}.

3.2 Co-Memorizing

We observed that not all progressions in the chain (*i.e.,* a single round of software optimization) lead to better solutions. This includes *solution backtracking*, where optimization loops back to earlier content, and *correct-to-failure degeneration*, where functioning software is inadvertently changed into a unexecutive solution. These scenarios suggest some steps in the process are redundant or ineffective, indicating that solely memorizing complete historical trajectories may be insufficient for designing agents' experiences. Thus, we convert the chain into a *task execution graph* to map nodes with the same content in the chain to a shared node in a redefined ($\leftarrow$) graph:

$$\mathcal{N} \leftarrow \{\phi(s_j) | s_j \in \mathcal{N}\}$$
$$\mathcal{E} \leftarrow \{(\phi(s_j), i_{j+1}, \phi(s_{j+1})) | (s_j, i_{j+1}, s_{j+1}) \in \mathcal{E}\}$$

where ϕ is a mapping rule using a hash function (Sasaki and Aoki, 2009). This approach efficiently groups identical solutions and highlights repetitions, serving as a "state transition graph" throughout the task execution process, as visualized in Figure 1.

Furthermore, in a task execution process, each edge linking two adjacent nodes signifies one round of autonomous solution optimization by the agents, indicating that the agents have already possessed the corresponding decision-making capability for each existing edge. Hence, relying solely on existing edges might not suffice for the design of agents' past experiences. For this purpose, the co-memorizing module is devised to heuristically identify shortcuts linking non-adjacent nodes on the graph. These form the basis of the agents' experience pools in practical reasoning, with the goal of accelerating future task execution.

Node Estimation Firstly, the score of each node s_j in $\mathcal{N}$ is estimated using an external feedback signal, calculated in a pairwise manner:

$$\omega(s_j) = sim(s_j, task) \times sim(s_j, s_{|\mathcal{N}|}) \times [\![s_j]\!]$$

where $sim(\cdot, \cdot)$ calculates the similarity between a node with another node or a task requirement, achieved through the use of an external code embedder and a text embedder, while $[\![\cdot]\!]$ indicates a binary signal indicating whether compilation is successful via an external compiler. The heuristic scoring rule assigns high evaluations to solutions that meet the task requirements, resemble the final solution, and are validated by an external tool.

Shortcut Extraction To discover informative shortcuts, we selectively identify shortcuts linking non-adjacent nodes that exceed an information gain threshold ϵ:

$$\mathsf{S} = \{(s_i, \overrightarrow{s_i s_j}, s_j) | s_i, s_j \in \bar{\mathcal{N}} \wedge (s_i, \cdot, s_j) \notin \mathcal{E} \wedge [\![s_i \rightarrow s_j]\!] \wedge \omega(s_j) - \omega(s_i) \geq \epsilon\}$$

where $\bar{\mathcal{N}}$ represents the nodes on the shortest path between the source node s_0 (*i.e.,* empty solution) and the sink node $s_{|\mathcal{N}|}$ (*i.e.,* final solution) in $\mathcal{N}$, $[\![s_i \rightarrow s_j]\!]$ indicates that s_j is reachable from s_i. Since the instruction of each shortcut edge does not inherently exist in the trajectory, for the two non-adjacent nodes connected by the shortcut, we create a pseudo instruction $\overrightarrow{s_i s_j}$ that effectively links the two non-adjacent nodes using a standard self-instruction mechanism (Wang et al., 2023e). This involves creating an instruction by comparing two distinct codes. This mechanism only extracts informative shortcuts on the shortest path and integrates compilation signals, naturally mitigating redundant solution backtracking and unexpected correct-to-failure degeneration.

Experience Gathering To leverage the heuristically-discovered shortcuts identified from historical trajectories as past experiences, the agents accumulate their own experience pools with key-value pairs for the future reasoning:

$$\mathsf{S}_I = \bigcup_{\forall t \in \mathcal{T}} \{(s_i, \overrightarrow{s_i s_j}) | (s_i, \overrightarrow{s_i s_j}, s_j) \in \mathsf{S}_t\}$$
$$\mathsf{S}_A = \bigcup_{\forall t \in \mathcal{T}} \{(\overrightarrow{s_i s_j}, s_j) | (s_i, \overrightarrow{s_i s_j}, s_j) \in \mathsf{S}_t\}$$

where S_t denotes the shortcut set extracted from a task t and $\mathcal{T}$ is the whole training set. This signifies that the instructor retains solution-to-instruction experiences to refine its instructional capabilities, while the assistant preserves instruction-to-solution experiences to enhance solution generation.

The design of shortcuts-as-experiences allows for an escape from the solution optimization capabilities that agents have already possessed in each step of their historical execution process, providing the possibility for more efficient shortcut-driven "accelerated" reasoning.

3.3 Co-Reasoning

The co-reasoning module is designed to combine the collective experience pools of agents, enabling communication through augmented instructions and solutions. By leveraging their respective experiential knowledge, these agents access and generate more refined answers, enhancing their collaborative task-solving abilities on unseen tasks.

The process begins with the instructor, armed with a solution-to-instruction memory S_I, encountering the current task solution s_j. It starts by using a retrieval tool to access experiential instructions that closely match the latent meaning of the query (Lewis et al., 2020). These instructions serve as few-shot examples (Zhao et al., 2024; Rubin et al., 2022; Min et al., 2022), guiding the instructor's reasoning to produce i^*_{j+1}, which is then shared with the assistant. The assistant, equipped with an instruction-to-solution memory S_A, retrieves optimal solutions based on the received instruction. These solutions form the foundation for the assistant's few-shot examples, culminating in the formulation of a new solution s^*_{j+1}. This entire procedure can be represented as:

$$i^*_{j+1} = \mathbb{I}(s_j, \Bbbk(s_j, \mathsf{S}_I)) \quad s^*_{j+1} = \mathbb{A}(s_j, \Bbbk(i^*_{j+1}, \mathsf{S}_A))$$

where $\Bbbk(q, s)$ denotes the retrieval of top-k matched results using q as a query in a key-value database s, $\mathbb{I}$ and $\mathbb{A}$ are the in-context reasoning functions of the instructor and assistant, respectively, utilizing few-shot examples.

In each communication, the solution obtained is used as the next step in the agents' ongoing communication. This process for each task is represented as a sequence of pairs $\{(i^*_1, s^*_1), (i^*_2, s^*_2), \cdots\}$, where each pair includes an experience-enhanced instruction and the corresponding solution.

4 Evaluation

Baselines We chose different types of LLM-driven software development paradigms as our baselines, which include both single-agent and multi-agent methodologies. GPT-Engineer (Osika, 2023) is a foundational single-agent approach in the evolving domain of LLM-powered software agents; its standout feature is its exceptional proficiency in accurately comprehending input task requirements and employing one-step reasoning, which significantly enhances its efficiency in producing comprehensive software solutions at the repository level. MetaGPT (Hong et al., 2024) 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; within each substep, solutions are generated through a single-step solution by agents with varying capabilities. ChatDev (Qian et al., 2023) is an LLM-powered agent collaborative software development framework that organizes the entire software development process into waterfall-style phases (*e.g.,* code completion, code review, and system testing); within this framework, software agents engage in task-oriented and multi-turn communications that play a pivotal role in enhancing software development quality by iteratively providing instructions and solutions during their communications.

Datasets We leveraged the SRDD dataset (Qian et al., 2023), which contains various software requirement descriptions. This dataset, reflecting categories from major software store platforms, was crafted to minimize redundancy while enhancing originality and diversity. It consists of 1,200 software requirements, systematically arranged into 5 primary categories (`Education`, `Work`, `Life`, `Game`, and `Creation`). These main categories are segmented into 40 distinct subcategories, with each subcategory containing 30 unique tasks.

Metrics Evaluating software is a challenging task, especially when trying to assess it on a holistic level. Traditional metrics like function-level code evaluation (e.g., `pass@k`) cannot seamlessly transfer to a comprehensive evaluation of entire software systems. This is primarily because it's often impractical to create manual or automated test cases for much of the software, particularly when dealing with complex interfaces, frequent communications, or non-deterministic feedback. As a solution, we use three quantifiable and objective dimensions to assess specific aspects of the software, and then

Method	Paradigm	Completeness	Executability	Consistency	Quality	Duration (s)
GPT-Engineer	🤓	0.4824	0.3583	0.7887	0.1363	**15.6000**
MetaGPT	🤓🤓	0.4472	0.4208	0.7649	0.1439	154.0000
ChatDev	🤓🤓	0.6131	0.8800	0.7909	0.4267	148.2150
Co-Learning	🤓🤓	**0.9497**	**0.9650**	**0.7970**	**0.7304**	122.7750

Table 1: Overall performance of the representative software development methods, encompassing both single-agent (🤓) and multi-agent (🤓🤓) paradigms. Performance metrics are averaged across all tasks in the test set. The **highest** scores are formatted in bold, while the second-highest scores are underlined.

combine these dimensions into a comprehensive metric for a more holistic evaluation:

- *Completeness* measures the software's ability to fulfill code completion in software development, quantified as the percentage of software without any "TODO" code snippets. A higher score indicates a higher probability of automated execution.

- *Executability* assesses the software's ability to run correctly within a compilation environment, quantified as the percentage of software that compiles successfully and can run directly. A higher score indicates a higher probability of successful execution.

- *Consistency* 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* is a comprehensive metric that integrates the aspects of completeness, executability, and consistency to assess the overall quality of software.[2] A higher quality score suggests a higher overall quality of the software generated, implying a lower need for further manual intervention.

Implementation Details Our system explicitly encompasses essential phases such as code complete, code review, and system testing. In the co-tracking phase, we integrate *GPT-3.5-Turbo* as the foundational model, limiting communication rounds between agents to a maximum of *5* per phase. For co-memorizing, we select *text-embedding-ada-002* as the semantic embedder due to its exceptional performance in both text and code embeddings. We utilize *MD5* as our hashing function, and *Python-3.11.4* is employed to provide environment feedback. Agents have access to relevant tools like code checkers and compilers. Additionally, it applies an information gain threshold of *0.90* to retain informative shortcuts. In the co-reasoning module, this approach selects the highest-ranked result from code and text experience pools as the in-context example for reasoning. Besides, we divided the dataset into training, validation, and testing sets in a 4:1:1 ratio, using random hierarchical sampling to maintain a balanced category split. The training set is utilized for co-tracking and co-memorizing to gather experiences, the validation set for choosing hyperparameters, and the test set for co-reasoning. To maintain comparability in experimental results, all baseline models use identical parameters and environment settings.

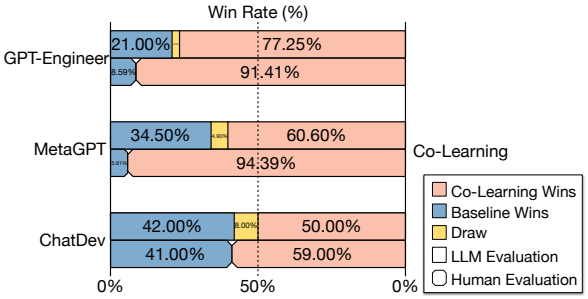

Figure 2: GPT-4 and human as the evaluators for pairwise comparisons of the solutions generated.

4.1 Quality Analysis

According to Table 1, our method, abbreviated as Co-Learning, significantly outperforms all three established baseline models in terms of quality. The comparison with ChatDev, a powerful multi-agent framework, is particularly striking, as Co-Learning significantly boosts the comprehensive metric from

[2]To prevent over-complication, the quality metric is defined through the multiplication of completeness, executability, and consistency. Employing a simple average sum would produce similar results and conclusions.

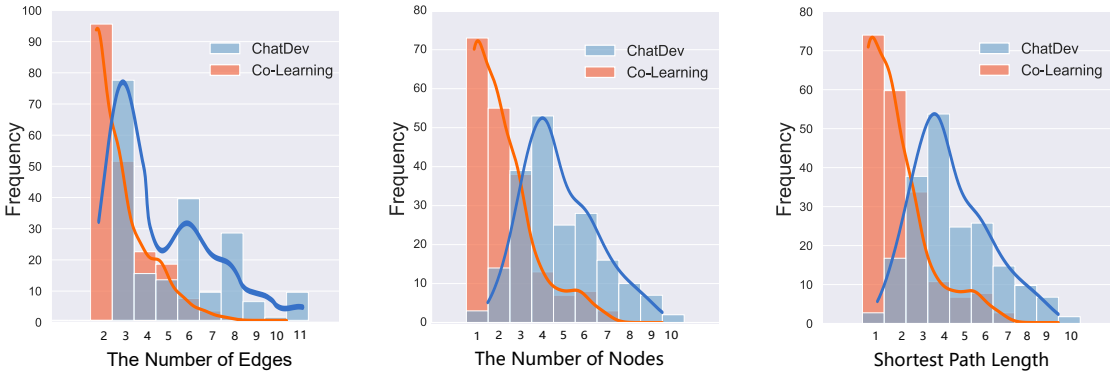

Figure 3: Distribution of key elements in task execution graphs. In a task execution graph, the number of edges represents the total communications between agents during code iterations in software development. The nodes count reflects the unique software source codes after hash deduplication, indicating the solution space during software optimization. The shortest path length shows the main path length in the task development process, excluding cycles and invalid attempts.

0.4267 to 0.7304, highlighting the effectiveness of utilizing agents' past experiences in addressing unseen tasks. The efficacy of our method primarily stems from the agents' proficiency in recalling and applying high-quality shortcuts from past trajectories, leading to notable enhancements in key performance metrics like completeness and executability for unseen tasks.

Moreover, Co-Learning outperforms GPT-Engineer, illustrating the superiority of the multi-agent approach in segmenting task-solving into discrete subtasks, in contrast to a single-step strategy. Through active communication, each agent contributes to a dynamic collaboration, steering towards cohesive and automated solutions for task execution. The effectiveness of this approach in task decomposition is further validated by contemporary studies (Wei et al., 2022b). Additionally, although Co-Learning requires more time compared to single-agent methods, it proves to be more time-efficient than other multi-agent approaches, suggesting that while multi-agent communications inherently take longer, the strategic use of "shortcut" patterns from past experiences effectively reduces reasoning time, striking a balance between performance and duration.

Upon closer examination, while Co-Learning exhibits notable strengths in the areas of completeness and executability, it demonstrates only a slight enhancement in the aspect of consistency. This result could likely stem from the embedding models' broad-grained semantic representation capabilities for text and code, which might not be adequately sensitive to discern extremely nuanced inconsistencies. This discovery presents an exciting research opportunity to develop advanced criteria and metrics for assessing software's consistency with its text requirements, emphasizing the need for more refined evaluation methodologies.

To further validate the efficacy of our method from an alternative perspective, we drew inspiration from previous work (Li et al., 2023a), which adopts both GPT-4 and human participants for pairwise comparisons of the solutions producted by agents.[3] Figure 2 illustrates that our Co-Learning method consistently surpasses other baseline methods, as evidenced by its higher average win rates in both evaluations involving GPT-4 and human participants. The results also reveal that ChatDev consistently demonstrates a notably high win rate compared to other baseline methods, indicating its strength as a robust baseline. This success is largely due to ChatDev's approach of performing detailed task decomposition and software optimization via multi-round agent communications, effectively addressing potential shortcomings. The Co-Learning approach, akin to a ChatDev variant emphasizing experiential agents, highlights the importance of agent experience accumulation.

4.2 Efficiency Analysis

Recall that we explicitly construct a task execution graph in which shortcuts are heuristically extracted as agents' experiences. In this section, we delve

[3]For each task, GPT-4 evaluation compares solutions from both methods, avoiding positional bias (Wang et al., 2023c). In the human evaluation, thirty computer science researchers assessed the software, with solutions from both methods randomly presented for their preference-based selection.

deeper into the graph analysis to uncover fundamental patterns in the task execution process by agents. For comparison purposes, we selected ChatDev as it represents the strongest baseline currently available. Figure 3 shows that, compared to ChatDev, Co-Learning trends towards fewer numbers in terms of the number of edges, nodes, and the length of the shortest path. This suggests a decrease in the number of iterations required for software development and a simplification of the solution space during software optimization. The enhancement in efficiency can be largely credited to the strategic utilization of shortcuts linking non-adjacent nodes in the graph, which enables agents to leverage previous task execution experiences while simultaneously boosting their future task-solving skills more effectively through the adoption of "shortcut thinking". In conjunction with Table 1 and Figure 2, this evidence demonstrates that the Co-Learning method streamlines the software development process by decreasing unnecessary iterations, thereby not only boosting overall efficiency but also delivering higher quality solutions with fewer agent communication.

4.3 Effectiveness Analysis

In this part, we delve into the distinct roles of instructors and assistants within the agent-collaboration framework, focusing on scenarios where either one agent alone has past experiences or both agents are inexperienced. As illustrated in Table 2, relying solely on a single agent leads to a marked decline in overall performance, manifesting in an increase in execution communications and a decrease in the quality. Furthermore, it is notable that systems with an experienced instructor perform worse than those with an experienced assistant (0.5305 vs. 0.6840), indicating the greater significance of the assistant's role in task execution, despite both roles being essential. In situations where neither agent type has experience, the system reverts to its traditional technological capabilities, resulting in the least effective performance solutions. These findings highlight the necessity for both instructors and assistants to be experienced; systems lacking this or relying on only one experienced agent demonstrate diminished execution efficiency and quality.

To further confirm the effectiveness of the key mechanisms employed in our framework, we conduct experiments using three different configurations and the results are presented in Table 2, consistently exhibiting a decline in performance. The *adjacent-execution* variant adopts the experiences of adjacent nodes from the original trajectories (after successfully compilation) rather than utilizing shortcuts, equivalent to ExpeL (Zhao et al., 2024). This approach leads to a significant increase in the number of experiences, ultimately resulting in a decrease in the quality of experiences. This, in turn, validates the effectiveness of the proposed "shortcuts-as-experiences" scheme. The *longest-shortcut-only* variant discards all intermediate shortcuts and exclusively relying on the "longest" shortcut that directly connects the start and end nodes. It reveals the underlying models' limitations and its inability to handle new tasks using only single-step historical experiences, indicating an overstretch of the agent's contextual reasoning ability. The *graph-unconstructed* variant extracts shortcuts directly from the original trajectory without constructing graphs. The obtained solution also reveals that over-dependence on past experience fails to achieve optimal performance, as it creates many shortcuts for possible graph nodes, leading to unnecessary repetition in the test set and decreasing task execution efficiency. This result confirms our observation that the progression of arbitrary adjacent nodes does not necessarily lead to continuously improved solutions. This emphasizes the significance of using heuristic shortcuts in deduplicated graphs to strike a balance between quantity and quality.

Method	#Experiences	#Nodes	#Edges	Quality
Co-Learning	(537, 537)	**2.3100**	**3.0100**	**0.7304**
⧹Instructor's Experiences	(0, 537)	3.3500	3.8850	0.6840
⧹Assistant's Experiences	(537, 0)	4.4422	5.0352	0.5305
⧹Both Experiences	(0, 0)	3.9450	4.7950	0.4267
↻Adjacent-Execution	(1604, 1604)	3.7000	4.5000	0.6398
↻Longest-Shortcut-Only	(332, 332)	2.8700	3.5200	0.6752
↻Graph-Unconstructed	(605, 605)	2.7000	3.4350	0.6821

Table 2: Ablation study on main roles or mechanisms of our framework. ↻ and ⧹ denote the replacement operation and the removing operation respectively. (a, b) indicates that the instructor and assistant are equipped with a and b heuristic shortcuts respectively.

4.4 Sensitivity Analysis

In this section, we explores the effects of two key parameters in the co-reasoning process: k (the number of matched results in retrieval) and θ (the semantic similarity threshold utilized in retrieval). The results obtained from retrieval come in two forms: text and code, so the number of matched

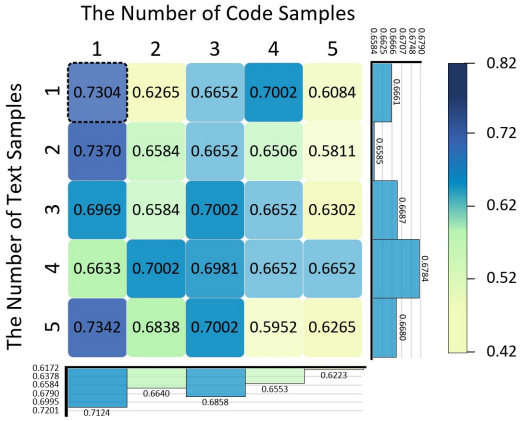

Figure 4: The effect of the top-k, with the dashed line indicating the default configuration utilized in the above experiments.

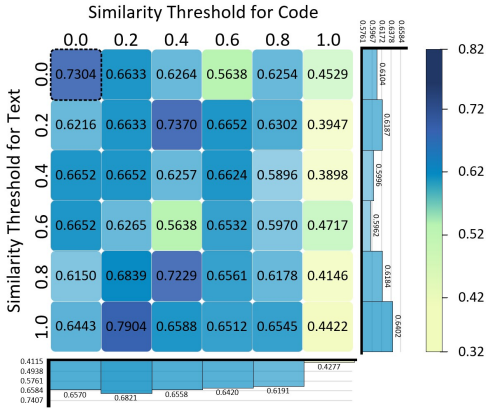

Figure 5: The effect of the semantic similarity threshold, with the dashed line indicating the default configuration utilized in the above experiments.

results are denoted as k_{text} and k_{code}, ranging from 1 to 5, and the semantic similarity thresholds are denoted as θ_{text} and θ_{code}, ranging from 0.0 to 1.0 in increments of 0.20.

Figure 4 shows that the best retrieval performance is achieved under the configuration ($k_{code} = 1, k_{text} = 2$), surpassing the default setting ($k_{code} = 1, k_{text} = 1$). This indicates that although the proposed framework has already achieved success under its default setting, there is still potential for further improvement through hyperparameter optimization. Additionally, Figure 4 suggest that optimizing hyperparameters related with code yields superior outcomes than optimizing hyperparameters related with text, aligning well with the conclusions from our previous experiments. Figure 5 illustrates that the θ parameter has a limited impact on the results, which is attributed to the consistently high semantic similarity between the current query and the retrieval result, often surpassing 0.85.

5 Conclusion

Recognizing the absence of a mechanism for integrating cumulative experiences from past tasks in agent collaboration, we have proposed Experiential Co-Learning, a framework that encourages collaboration between autonomous agents. Through co-tracking, co-memorizing, and co-reasoning, this approach enables agents to efficiently handle unseen software development tasks by drawing on past experiences and providing mutual support. The quantitative analysis effectively showcased significant improvements in quality, leading to reduced execution times, decreased repetitive errors, and a decreased reliance on additional human intervention. We anticipate that our insights will initiate a paradigm shift in shaping the design of multi-agent collaboration, propelling agents towards achieving greater autonomy and contributing to their evolutionary growth in cooperative learning.

6 Limitations

Our study has explored the capabilities of cooperative autonomous agents, yet both researchers and practitioners should be mindful of certain limitations and risks. Firstly, the ability of autonomous agents to produce software might be overestimated. In co-learning, autonomous agents still tend to implement the simplest logic in absence of necessary and clear requirements, indicating that these technologies are more suitable for prototype systems rather than real-world applications. Secondly, unlike traditional function-level code generation approach, automating the evaluation of general-purpose software is exceptionally challenging. Though recent efforts aimed to employ *Human Revision Cost* (Hong et al., 2024), manual verification is still impractical. This paper instead focuses on three objective and crucial dimensions and a comprehensive quality. Future research need take additional dimensions like robustness into consideration.

Acknowledgments

The work was supported by the National Key R&D Program of China (No.2022ZD0116312), the Postdoctoral Fellowship Program of CPSF under Grant Number GZB20230348, and Tencent Rhino-Bird Focused Research Program.

References

Henri Barki, Suzanne Rivard, and Jean Talbot. 1993. Toward an Assessment of Software Development Risk. In *Journal of Management Information Systems*, volume 10, pages 203–225.

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

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 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. In *arXiv preprint arXiv:2303.12712*.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2024. Large Language Models as Tool Makers. In *The Twelfth International Conference on Learning Representations (ICLR)*.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2024. Chateval: Towards Better LLM-based Evaluators through Multi-agent Debate. In *The Twelfth International Conference on Learning Representations (ICLR)*.

Dake Chen, Hanbin Wang, Yunhao Huo, Yuzhao Li, and Haoyang Zhang. 2023. GameGPT: Multi-agent Collaborative Framework for Game Development. In *arXiv preprint arXiv:2310.08067*.

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

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chen Qian, Chi-Min Chan, Yujia Qin, Yaxi Lu, Ruobing Xie, et al. 2024. Agentverse: Facilitating Multi-agent Collaboration and Exploring Emergent Behaviors in Agents. In *The Twelfth International Conference on Learning Representations (ICLR)*.

Roi Cohen, May Hamri, Mor Geva, and Amir Globerson. 2023. LM vs LM: Detecting Factual Errors via Cross Examination. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 12621–12640.

Katherine Compton and Scott Hauck. 2002. Reconfigurable Computing: a Survey of Systems and Software. In *ACM Computing Surveys (csuR)*, volume 34, pages 171–210.

Shiying Ding, Xinyi Chen, Yan Fang, Wenrui Liu, Yiwu Qiu, and Chunlei Chai. 2023. DesignGPT: Multi-Agent Collaboration in Design. In *2023 16th International Symposium on Computational Intelligence and Design (ISCID)*, pages 204–208.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. 2020. Shortcut Learning in Deep Neural Networks. In *Nature Machine Intelligence*, volume 2, pages 665–673.

Ran Gong, Qiuyuan Huang, Xiaojian Ma, Hoi Vo, Zane Durante, Yusuke Noda, Zilong Zheng, Song-Chun Zhu, Demetri Terzopoulos, Li Fei-Fei, and Jianfeng Gao. 2023. MindAgent: Emergent Gaming Interaction. In *arXiv preprint arXiv:2309.09971*.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In *The Twelfth International Conference on Learning Representations (ICLR)*.

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

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2023. Benchmarking Large Language Models as AI Research Agents. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

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

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020.

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 9459–9474.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023a. CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society. In *Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS)*.

Yuan Li, Yixuan Zhang, and Lichao Sun. 2023b. Metaagents: Simulating Interactions of Human Behaviors for LLM-based Task-oriented Coordination via Collaborative Generative Agents. In *arXiv preprint arXiv:2310.06500*.

Weixin Liang, Yuhui Zhang, Hancheng Cao, Binglu Wang, Daisy Ding, Xinyu Yang, Kailas Vodrahalli, Siyu He, Daniel Smith, Yian Yin, Daniel McFarland, and James Zou. 2023. Can Large Language Models Provide Useful Feedback on Research Papers? A Large-Scale Empirical Analysis. In *arXiv preprint arXiv:2310.01783*.

Zhiwei Liu, Weiran Yao, Jianguo Zhang, Le Xue, Shelby Heinecke, Rithesh Murthy, Yihao Feng, Zeyuan Chen, Juan Carlos Niebles, Devansh Arpit, Ran Xu, Phil Mui, Huan Wang, Caiming Xiong, and Silvio Savarese. 2023. BOLAA: Benchmarking and Orchestrating LLM-augmented Autonomous Agents. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Kaixin Ma, Hongming Zhang, Hongwei Wang, Xiaoman Pan, and Dong Yu. 2023. LASER: LLM agent with state-space exploration for web navigation. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

Harlan D Mills. 1976. Software development. In *IEEE Transactions on Software Engineering*, 4, pages 265–273.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the Role of Demonstrations: What Makes In-Context Learning Work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 11048–11064.

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

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training Language Models to Follow Instructions with Human Feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 27730–27744. Curran Associates, Inc.

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 (UIST)*, pages 1–22.

Chen Qian, Xin Cong, Wei Liu, Cheng Yang, Weize Chen, Yusheng Su, Yufan Dang, Jiahao Li, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. ChatDev: Communicative Agents for Software Development. In *Proceedings of the 62st Annual Meeting of the Association for Computational Linguistics (ACL)*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2024. Toolllm: Facilitating Large Language Models to Master 16000+ Real-World APIs. In *The Twelfth International Conference on Learning Representations (ICLR)*.

Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, Xuanhui Wang, and Michael Bendersky. 2023. Large Language Models are Effective Text Rankers with Pairwise Ranking Prompting. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language Models are Unsupervised Multitask Learners. In *OpenAI blog*, volume 1, page 9.

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

Jingqing Ruan, YiHong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, du qing, shi shiwei, Hangyu Mao, Xingyu Zeng, and Rui Zhao. 2023. TPTU: Task Planning and Tool Usage of Large Language Model-based AI Agents. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2022. Learning To Retrieve Prompts for In-Context Learning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, pages 2655–2671.

Yu Sasaki and Kazumaro Aoki. 2009. Finding Preimages in Full MD5 Faster Than Exhaustive Search. In *Advances in Cryptology - EUROCRYPT 2009*, pages 134–152.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS)*.

Murray Shanahan, Kyle McDonell, and Laria Reynolds. 2023. Role Play with Large Language Models. In *Nature*, volume 623, pages 493–498.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Theodore Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas Griffiths. 2024. Cognitive architectures for language agents. In *Transactions on Machine Learning Research (TMLR)*. Survey Certification.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023a. Voyager: An Open-Ended Embodied Agent with Large Language Models. In *Intrinsically-Motivated and Open-Ended Learning Workshop @NeurIPS2023*.

Lei Wang, Jingsen Zhang, Hao Yang, Zhiyuan Chen, Jiakai Tang, Zeyu Zhang, Xu Chen, Yankai Lin, Ruihua Song, Wayne Xin Zhao, Jun Xu, Zhicheng Dou, Jun Wang, and Ji-Rong Wen. 2023b. When Large Language Model based Agent Meets User Behavior Analysis: A Novel User Simulation Paradigm. In *arXiv preprint arXiv:2306.02552*.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. 2023c. Large Language Models are not Fair Evaluators. In *arXiv preprint arXiv:2305.17926*.

Shenzhi Wang, Chang Liu, Zilong Zheng, Siyuan Qi, Shuo Chen, Qisen Yang, Andrew Zhao, Chaofei Wang, Shiji Song, and Gao Huang. 2023d. Avalon's Game of Thoughts: Battle Against Deception through Recursive Contemplation. In *Findings of the Association for Computational Linguistics: ACL 2024*.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P. Xing, and Zhiting Hu. 2024. PromptAgent: Strategic Planning with Language Models Enables Expert-level Prompt Optimization. In *The Twelfth International Conference on Learning Representations (ICLR)*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023e. Self-Instruct: Aligning Language Models with Self-Generated Instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 13484–13508.

Zhilin Wang, Yu Ying Chiu, and Yu Cheung Chiu. 2023f. Humanoid Agents: Platform for Simulating Human-like Generative Agents. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (EMNLP)*, pages 167–176.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022b. Chain-of-thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 24824–24837.

Lilian Weng. 2023. LLM-powered Autonomous Agents. In *lilianweng.github.io*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2023. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework. In *arXiv preprint arXiv:2308.08155*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2024. Large Language Models as Optimizers. In *The Twelfth International Conference on Learning Representations (ICLR)*.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2023. GPT4Tools: Teaching Large Language Model to Use Tools via Self-instruction. In *Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS)*.

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

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. ExpeL: LLM Agents Are Experiential Learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 17, 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. 2024. Webarena: A realistic Web Environment for Building Autonomous Agents. In *The Twelfth International Conference on Learning Representations (ICLR)*.

Wangchunshu Zhou, Yuchen Eleanor Jiang, Long Li, Jialong Wu, Tiannan Wang, Shi Qiu, Jintian Zhang, Jing Chen, Ruipu Wu, Shuai Wang, Shiding Zhu, Jiyu Chen, Wentao Zhang, Ningyu Zhang, Huajun Chen, Peng Cui, and Mrinmaya Sachan. 2023. Agents: An Open-source Framework for Autonomous Language Agents. In *arXiv preprint arXiv:2309.07870*.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, Yu Qiao, Zhaoxiang Zhang, and Jifeng Dai. 2023. Ghost in the Minecraft: Generally Capable Agents for Open-World Environments via Large Language Models with Text-based Knowledge and Memory. In *arXiv preprint arXiv:2305.17144*.