Experiential Co-Learning of Software-Developing Agents

Chen Qian[†♠] **Yufan Dang**[†♠] **Jiahao Li**[♦] **Wei Liu**[♠]
Weize Chen[♠] **Cheng Yang**[♣✉] **Zhiyuan Liu**[♠✉] **Maosong Sun**[♠✉]
[♠]Tsinghua University [♦]Dalian University of Technology
[♣]Beijing University of Posts and Telecommunications
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. These agents are now capable of collaborating seamlessly, splitting tasks and enhancing accuracy, thus minimizing the need for human involvement. However, these agents often approach a diverse range of tasks in isolation, without benefiting from past experiences. This isolation can lead to repeated mistakes and inefficient trials in task solving. To this end, this paper introduces *Experiential Co-Learning*, a novel framework in which instructor and assistant agents gather shortcut-oriented experiences from their historical trajectories and use these past experiences for mutual reasoning. This paradigm, enriched with previous experiences, equips agents to more effectively address unseen tasks.

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 use of external tools (Schick et al., 2023). This enhancement has expanded their capacity to effectively manage a broader spectrum of complex tasks, including mathematical reasoning (Wei et al., 2022b; Lu et al., 2023), software development (Osika, 2023; Qian et al., 2023), game playing (Wang et al., 2023a; Zhu et al., 2023; Wang et al., 2023c; Gong et al., 2023), social simulation (Park et al., 2023; Li et al., 2023b; Wang et al., 2023f; Hua et al., 2023), and scientific research (Huang et al., 2023; Liang et al., 2023). In this paper, we take software development (Mills, 1976) as an representative scenario, chosen due to its complexity that demands a blend of natural and programming language skills, the continuity that often requires an in-depth understanding of coding and continuous alterations (Mills, 1976; Abrahamsson et al., 2017; Barki et al., 1993), and the clarity of code that can provide quantifiable measures (Compton and Hauck, 2002).

With the development of autonomous-agent technology, a successful breakthrough has been the integration of interaction among multiple agents (Park et al., 2023; Li et al., 2023a; Qian et al., 2023). Representative methods epitomize this methodology by segmenting task solving into distinct subtasks, during which agents adopt two types of roles—an instructor and an assistant. Through engaging in interactive dialogues, each agent participates in instructive and responsive conversations, collaboratively contributing to the achievement of a cohesive and automated solution for task completion. The development of a more adaptive and proactive approach to problem-solving by these agents marks a significant leap in autonomy, going beyond the typical prompt-guided dynamic in human-computer interactions (Yang et al., 2023a) and substantially reducing dependence on human involvement (Li et al., 2023a; Qian et al., 2023; Wu et al., 2023). For example, during the automated software development process in ChatDev (Qian et al., 2023), a reviewer agent iteratively provides instructions for software optimization (*e.g.,* code completion, functional optimization, and program debugging), to which a programmer agent responds with updated source codes.

However, when confronted with a diverse

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

range of task types, agents tend to carry out each task independently. A notable drawback in this methodology is the lack of a mechanism for integrating the cumulative experiences gained from previous tasks (Qian et al., 2023; Chen et al., 2023b; Park et al., 2023; Hong et al., 2023). Furthermore, the inexperience nature through multi-step task execution frequently results in repetitive errors or unnecessary trial-and-error processes in task solving, ultimately necessitating additional human involvement (Li et al., 2023a).

To tackle these challenges, we propose *Experiential Co-Learning*, a novel multi-agent learning paradigm designed to enhance agents' interactive task-solving abilities by leveraging accumulated experiences. It integrates the two types of language agents through three specialized modules: the *co-tracking* module promotes interactive rehearsals between the agents, fostering joint exploration and the creation of procedural trajectories for various training tasks; the *co-memorizing* module heuristically mines "shortcuts" from historical trajectories using external environment feedback, which are then preserved in their experience pools in an interleaved manner; the *co-reasoning* module encourages agents to enhance their instructions and responses 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.

To summarize, we have made the following contributions:

- To our knowledge, this study is the first to integrate past experiences into the collaboration of LLM-powered agents. Through co-tracking, co-memorizing, and co-reasoning, this framework enables two types of cooperative agents (instructor and assistant) to efficiently handle software development by leveraging past experiences extracted from procedural trajectories.

- We suggest constructing execution graphs based on procedural trajectories, in which "shortcuts" that are not directly-connected are documented as experiences, which can effectively motivate agents to engage in shortcut thinking during every interaction.

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

2 Related Work

Large language models (LLMs), trained on vast datasets to comprehend and manipulate billions of parameters, have become pivotal in natural language processing (Brown et al., 2020; Bubeck et al., 2023; Vaswani et al., 2017; Radford et al.; 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., 2023a; Qin et al., 2023b; Kaplan et al., 2020). Recent progress, particularly in the field of autonomous agents (Zhou et al., 2023a; Wang et al., 2023a; Park et al., 2023; Wang et al., 2023f; Richards, 2023; Osika, 2023; Wang et al., 2023d), 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., 2023), planning (Chen et al., 2023b; Liu et al., 2023) and tool use (Schick et al., 2023; Cai et al., 2023; Qin et al., 2023a; Ruan et al., 2023; Yang et al., 2023b), enabling them to operate independently in complex, real-world scenarios (Zhao et al., 2023; Zhou et al., 2023a; Ma et al., 2023; Zhang et al., 2023; Wang et al., 2023b; Ding et al., 2023; Weng, 2023). For instance, GPT-Engineer (Osika, 2023) represents a leap in AI-assisted coding, competently constructing complete codebases from project outlines, customizing outputs to user preferences, and providing clarifications when necessary.

Parallel to these attempts, the autonomous interaction among multiple agents is now a promising paradigm, signaling a shift towards multi-agent task solving paradigm (Li et al., 2023a; Park et al., 2023; Zhou et al., 2023b; Chen et al., 2023b; Chan et al., 2023; Chen et al., 2023a; Cohen et al., 2023; Li et al., 2023b; Hua et al., 2023). These systems, featuring two distinct roles - an instructor and an assistant - enable collaborative interactions between autonomous agents to break down complex tasks into finer-grained subtasks (Qian et al., 2023; Hong et al., 2023; Wu et al., 2023). The instructor issues directional instructions, and the assistant provides relevant responses, facilitating a streamlined workflow for task completion. This approach not only boosts productivity but also exhibits a degree of

autonomy that surpasses the traditional prompt-guided paradigm in human-computer interactions, greatly minimizing the need for human involvement (Li et al., 2023a; Qian et al., 2023; Chen et al., 2023b). A notable example is ChatDev (Qian et al., 2023), a virtual software company powered by LLMs, which utilizes agents in roles such as reviewer and programmer within a chat-chain workflow, substantially improving the efficiency of software development procedures.

3 Experiential Co-Learning

Traditional multi-step interactions among autonomous agents 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; Li et al., 2023a; Wu et al., 2023; Park et al., 2023; Zhou et al., 2023b; Chen et al., 2023b; Chan et al., 2023; Chen et al., 2023a; Cohen et al., 2023; Li et al., 2023b). To remedy this, we propose *Experiential Co-Learning*, illustrated in Figure 1, powered by two distinct language agents and comprising three essential modules: the *co-tracking* module establishes a rehearsal collaboration between an instructor and an assistant, focusing on tracking their joint "procedural trajectories" for various training tasks, showcasing clear strategies in their interactive collaboration; the *co-memorizing* module strategically extracts "shortcuts" from the trajectories based on external environmental feedback, integrating these strategic paths into their collective experience pools; the *co-reasoning* module combines agents' collective experience pools to foster an advanced interaction of instructions and responses, improving their ability to collaboratively solve unseen tasks.

3.1 Co-Tracking

The co-tracking module sets up a collaborative rehearsal between an instructor and an assistant, with the goal of meticulously tracking "historical trajectories" for diverse training tasks (Mirchandani et al., 2023; Qin et al., 2023a). Each trajectory captures the detailed progression of a specific task, detailing the evolving interactions and clearly illustrating the strategies employed throughout the task completion process.

Formally, in the set of training tasks, each task fosters a functional interplay of interactions between agents, streamlined towards the effective completion of the task. During this process, the instructor provides a set of instructions $\mathcal{I} = \{i_1, i_2, ..., i_n\}$, to which the assistant responds with a corresponding sequence of responses $\mathcal{R} = \{r_1, r_2, ..., r_n\}$. This interactive dynamic can be naturally modeled as a directed chain $\mathcal{G} = (\mathcal{N}, \mathcal{E})$:

$$\mathcal{N} = \{r_j | r_j \in \mathcal{R}\} \cup \{r_0\}$$
$$\mathcal{E} = \{(r_j, i_{j+1}, r_{j+1}) | r_j, r_{j+1} \in \mathcal{R}, i_{j+1} \in \mathcal{I}\}$$

where $\mathcal{N}$ represents the nodes corresponding to the responses (with r_0 denoting the initial, typically empty state), and $\mathcal{E}$ denotes the edges corresponding to the instructions. Each edge (r_j, i_{j+1}, r_{j+1}) illustrates the transition from one response r_j to the subsequent response r_{j+1}, guided by the instruction i_{j+1}. This task execution chain effectively represents the task completion process, encapsulating the collaborative dynamics between both agents.

3.2 Co-Memorizing

Be aware that each step in the task execution process signifies a separate action performed by agents, steered by the underlying backbone models. In other words, the individual skills that agents currently possess are implicitly embedded into the models' parameters (Brown et al., 2020; Vaswani et al., 2017; Touvron et al., 2023). Hence, relying solely on adjacent nodes might not suffice for significant growth in experience. Bearing this in mind, the co-memorizing module is designed to efficiently discover non-adjacent "shortcuts" from procedural trajectories, subsequently integrating these into their collective experience pools. This mechanism allows the agents to reference past experiences, enhancing their future task-solving strategies with greater efficiency by employing shortcut thinking at each juncture.

Nonetheless, not all logical shortcuts in the execution process are of high quality. Our observations indicate that the progression of agents' task execution doesn't always progress towards increasingly superior outcomes (Qian et al., 2023; Petersen et al., 2009; Barki et al., 1993). Examples of this include version backtracking issues, where optimization processes sometimes loop back to previously developed versions, and correct-to-failure issues, where initially functioning software is accidentally altered into a version that fails to compile. These circumstances lead to certain steps in the entire execution process becoming redundant or ineffective.

To overcome the challenges, our approach involves transforming the task execution sequence into a graph, where nodes sharing the same versions

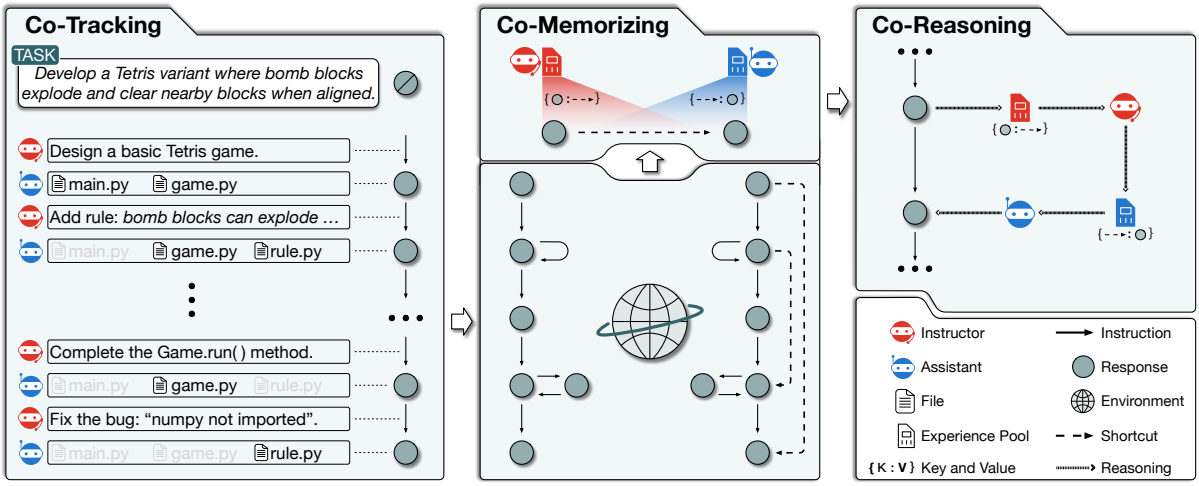

Figure 1: The framework of Experiential Co-Learning. The co-tracking module promotes interactive rehearsals between the agents, fostering joint exploration and the creation of procedural trajectories for various training tasks. The co-memorizing module strategically extracts "shortcuts" from the trajectories under external supervision, integrating these strategic paths into their collective experience pools. The co-reasoning module combines agents' collective experience pools to foster an advanced interaction of instructions and responses, improving their ability to collaboratively solve unseen tasks.

are clustered to discern their evolutionary roles. Then, non-adjacent nodes on the graph's shortest path are assessed under external supervision (*e.g.,* a compiler), leading to the discovery of shortcuts that could accelerate task completion. Finally, the proposed co-memorizing mechanism facilitates future task-solving by compelling the instructor to document the routes of discovered shortcuts for better guidance and the assistant to record their endpoints for better quality of corresponding responses.

Concretely, we begin by transforming the task execution chain $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ into a graph:

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

where ϕ is a rule that employs a hash function (Sasaki and Aoki, 2009) to map nodes in the chain to a common node in the graph. This mapping allows for the efficient aggregation of duplicate states and aids in tracking duplicate versions. Following this transformation, the value of each node r_j in $\mathcal{N}$ is estimated using an external feedback signal, calculated in a pairwise manner:

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

where $sim(\cdot)$ calculates the similarity between a node and the task description as well as the node and the terminal node, achieved through the use of an external code embedder and a text embedder, while $[\![\cdot]\!]$ indicates a binary compilation signal gained from an external compiler. The heuristic reward design ensures that responses aligned with the task requirements, similar to the final solution of a rehearsal trajectory, and successfully validated by an external tool, receive a notably high evaluation.

To discover potential shortcuts, we selectively identify pairs of non-adjacent nodes that exhibit an information gain exceeding a threshold ϵ:

$$\mathcal{M} = \{(r_i, \vec{r_i r_j}, r_j) | r_i, r_j \in \bar{\mathcal{N}} \wedge (r_i, \cdot, r_j) \notin \mathcal{E} \wedge [\![r_i \rightarrow r_j]\!] \wedge \omega(r_j) - \omega(r_i) \geq \epsilon\}$$

where $\bar{\mathcal{N}}$ denotes the nodes in the shortest path of $\mathcal{N}$, $[\![r_i \rightarrow r_j]\!]$ denotes that r_j is reachable from r_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 additionally generate a transition instruction $\vec{r_i r_j}$ that pseudo-connects the two nodes, implemented via self-instruct (Wang et al., 2023e). To leverage the heuristically-discovered shortcuts identified from various trajectories as past experiences, the agents encode their own memories with key-value pairs:

$$\mathcal{M}_I = \{(r_i, r_i \dashrightarrow r_j) | (r_i, r_i \dashrightarrow r_j, r_j) \in \mathcal{M}\}$$
$$\mathcal{M}_A = \{(r_i \dashrightarrow r_j, r_j) | (r_i, r_i \dashrightarrow r_j, r_j) \in \mathcal{M}\}$$

This signifies that the instructor retains response-to-instruction experiences to refine its instructional capabilities, while the assistant preserves instruction-to-response experiences to enhance solution generation. This co-memorizing mechanism promotes a synergistic evolution among the agents, enabling

them to engage in reciprocal reasoning when tackling future tasks.

3.3 Co-Reasoning

The co-reasoning module is designed to combine the collective experience pools of agents, enabling advanced interaction through instructions and responses. By leveraging their respective experiential knowledge, these agents access and generate more refined insights, enhancing their collaborative problem-solving abilities on unseen tasks.

The process begins with the instructor, armed with a response-to-instruction memory $\mathcal{M}_I$, encountering the current task state r_j. It starts by using a retrieval tool to access experiential instructions that closely match the meaning of the task (Lewis et al., 2021). These instructions serve as few-shot examples (Zhao et al., 2023; Rubin et al., 2021; 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-response memory $\mathcal{M}_A$, retrieves optimal responses based on the received instruction. These responses form the foundation for the assistant's few-shot examples, leading to the creation of a new response r^*_{j+1}. This entire procedure can be represented as:

$$i^*_{j+1} = \mathbb{I}(r_j, \Bbbk(r_j, \mathcal{M}_I))$$
$$r^*_{j+1} = \mathbb{R}(r_j, \Bbbk(i^*_{j+1}, \mathcal{M}_A))$$

where $\Bbbk(q, \mathcal{M})$ represents a retrieval operation that utilizes q as a query to fetch the top-k matched values from a key-value database $\mathcal{M}$, $\mathbb{I}(\cdot, e)$ and $\mathbb{R}(\cdot, e)$ represent in-context reasoning of the instructor and assistant, respectively, with e representing the few-shot examples.

At each interaction, the resulting response is applied as the subsequent state in the ongoing interaction between agents. Consequently, the entire process of agents' interactions for each task is represented as a series of pairs:

$$\{(i^*_1, r^*_1), (i^*_2, r^*_2), \cdots\}$$

Each pair in this sequence consists of an experience-augmented instruction from the instructor and the corresponding experience-augmented response from the assistant. This iterative process empowers the language agents to collectively enhance their task-solving capabilities by building upon prior states and accumulated experiences. The overall task completion process becomes notably more efficient through the integration of experiential knowledge.

4 Evaluation

In this section, we conduct a thorough comparison and evaluation of the proposed method across multiple dimensions, including autonomy, efficiency, effectiveness, and sensitivity, to ensure a comprehensive assessment.

4.1 Setup

Baselines In our comparative analysis, we chose different types of LLM-driven software development paradigms as our baselines, which include both single-agent and two-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 prompts and employing one-step reasoning, which significantly enhances its efficiency in producing comprehensive software solutions at the repository level. MetaGPT (Hong et al., 2023) is an innovative framework that assigns diverse roles to various LLM-powered agents and incorporates standardized operating procedures to facilitate multi-agent collaboration in software development; within each substep, solutions are generated through single-step decisions by agents with varying capabilities. ChatDev (Qian et al., 2023) is an LLM-powered multi-agent collaborative software development framework that organizes the entire software development process into waterfall-style phases (*e.g.,* code completion, code review, and integration testing); within this framework, software agents engage in task-oriented and multi-turn interactions that play a pivotal role in enhancing software development quality by iteratively providing instruction and responses during their interactions.

Datasets We leveraged the NLDD dataset (Qian et al., 2023), a specialized open-source collection designed for the "Natural Language to Software Generation" domain. This dataset, reflecting categories from major software store platforms, was meticulously crafted to minimize redundancy while enhancing originality and diversity. NLDD consists of 1,200 software task prompts, systematically arranged into 5 primary categories, such as `Education`, `Life`, and `Game`. These main categories are further segmented into 40 distinct subcategories, with each subcategory containing 30

Method	Paradigm	Completeness $\alpha \in [0,1]$	Executability $\beta \in [0,1]$	Consistency $\gamma \in [0,1]$	Autonomy $\alpha \times \beta \times \gamma \in [0,1]$	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	🤓🤓	<u>0.6131</u>	<u>0.6890</u>	<u>0.7909</u>	<u>0.3340</u>	148.2150
Co-Learning	🤓🤓	**0.9497**	**0.9400**	**0.7970**	**0.7100**	<u>122.7750</u>

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 <u>second-highest</u> scores are underlined.

unique tasks. We partitioned NLDD into training, validation, and testing sets using a 4:1:1 ratio. This segmentation, achieved through random hierarchical sampling, ensured a balanced representation across the various categories of the dataset.

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 interactions, or non-deterministic feedback. As a solution, we use three fundamental dimensions to assess specific aspects of the software, and then combine these dimensions into a comprehensive metric for a more holistic evaluation:

- *Completeness* (α) 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 completion.
- *Executability* (β) 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* (γ) 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.
- *Autonomy* ($\alpha \times \beta \times \gamma$) is a comprehensive metric that integrates the aspects of completeness, executability, and consistency to assess the overall quality of software. A higher autonomy score suggests a higher overall quality of the software generated, implying a lower need for further manual intervention.

Implementation Details Our system, developed using ChatDev, encompasses essential phases such as code review and software testing. In the co-tracking phase, we integrate `ChatGPT-3.5` as the foundational model, limiting interaction rounds between agents to a maximum of 5 per phase for structured engagement. For co-memorizing, we select `text-embedding-ada-002` 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. The process involves filtering experiences using an information gain threshold of 0.90 to ensure the retention of significant data. In the co-reasoning stage, our strategy focuses on retrieving only the most relevant and accurate result from both code and text searches, enhancing the precision of the semantic similarity retrieval. In addition, all baselines sample the same parameters and environment configuration to ensure the comparability of experimental results.

4.2 Autonomy Analysis

According to Table 1, our method, abbreviated as Co-Learning, significantly outperforms all three established baseline models in terms of autonomy, demonstrating a notable margin of improvement. The comparison with ChatDev, a powerful multi-agent framework, is particularly striking, as Co-Learning significantly boosts the comprehensive metric from 0.3340 to 0.7100, highlighting the effectiveness of utilizing agents'

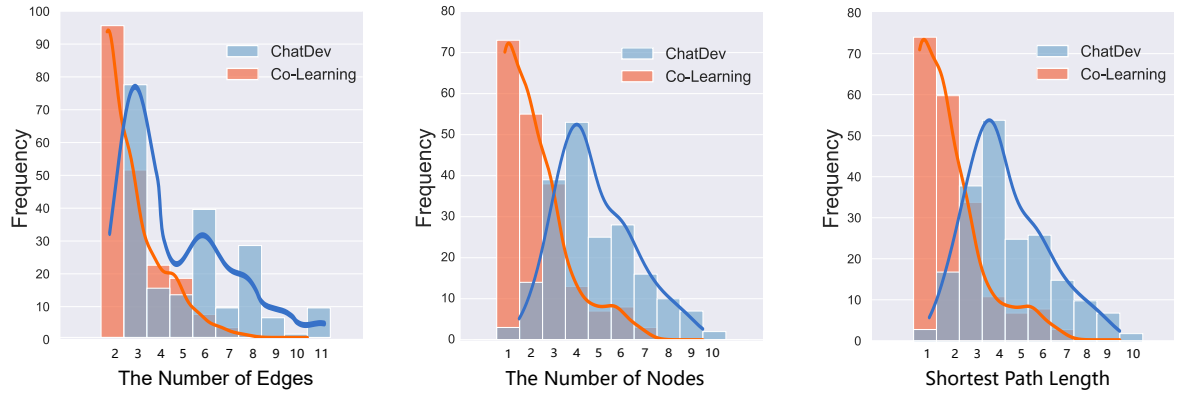

Figure 2: Distribution of key elements in task execution graphs. In a task execution graph, the number of edges represents the total interactions between agents during code iterations in software development. The nodes count reflects the unique software source codes after hash deduplication, indicating the software's state space. The shortest path length shows the main path length in the task development process, excluding cycles and invalid attempts.

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 experiences, which, combined with the underlying LLMs' "example imitation" ability, leads 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 problem-solving into discrete subtasks, in contrast to a single-step solution method. Through active interactive engagement, each agent contributes to a dynamic and insightful dialogue, collaboratively steering towards cohesive and automated solutions for task completion. The effectiveness of this approach in task decomposition is further corroborated 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 interactions 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 and comprehensive evaluation methodologies.

4.3 Efficiency Analysis

Recall that we explicitly construct a task execution graph for the whole process of agents working together to complete software development. In this section, we delve deeper into the graph analysis to uncover fundamental patterns in the task completion process by agents. For comparison purposes, we selected ChatDev as it represents the strongest baseline currently available. Figure 2 shows that in comparison with ChatDev, Co-Learning demonstrates a distinct shift towards the upper left in aspects like 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 software's state space. The enhancement in efficiency can be largely credited to the strategic utilization of shortcuts connecting 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, 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 outcomes with fewer agent iterations.

5 Conclusion

Recognizing a notable drawback in agents, the absence of a mechanism for integrating cumulative experiences from past tasks, 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 autonomy, leading to reduced execution times, decreased repetitive errors, and a decreased reliance on additional human intervention.

References

Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, and Juhani Warsta. 2017. Agile software development methods: Review and analysis. *arXiv preprint arXiv:1709.08439*.

Henri Barki, Suzanne Rivard, and Jean Talbot. 1993. Toward an assessment of software development risk. *Journal of management information systems*, 10(2):203–225.

Thorsten Brants, Ashok C Popat, Peng Xu, Franz J Och, and Jeffrey Dean. 2007. Large language models in machine translation.

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. Curran Associates, Inc.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4.

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

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

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

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

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

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

Katherine Compton and Scott Hauck. 2002. Reconfigurable computing: a survey of systems and software. *ACM Computing Surveys (csuR)*, 34(2):171–210.

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

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.

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. 2023. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework.

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.

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

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.

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. 2021. Retrieval-augmented generation for knowledge-intensive nlp tasks.

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

Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. 2023. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. In *International Conference on Learning Representations (ICLR)*.

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

Harlan D Mills. 1976. Software development. *IEEE Transactions on Software Engineering*, (4):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?

Suvir Mirchandani, Fei Xia, Pete Florence, Brian Ichter, Danny Driess, Montserrat Gonzalez Arenas, Kanishka Rao, Dorsa Sadigh, and Andy Zeng. 2023. Large language models as general pattern machines. *arXiv preprint arXiv:2307.04721*.

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

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. 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 Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback.

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.

Kai Petersen, Claes Wohlin, and Dejan Baca. 2009. The waterfall model in large-scale development. In *Product-Focused Software Process Improvement: 10th International Conference, PROFES 2009, Oulu, Finland, June 15-17, 2009. Proceedings 10*, pages 386–400. Springer.

Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. 2023. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*.

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

Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, Xuanhui Wang, and Michael Bendersky. 2023b. Large language models are effective text rankers with pairwise ranking prompting.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners.

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

Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi, Hangyu Mao, Ziyue Li, Xingyu Zeng, and Rui Zhao. 2023. Tptu: Large language model-based ai agents for task planning and tool usage.

Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2021. Learning to retrieve prompts for in-context learning. *arXiv preprint arXiv:2112.08633*.

Yu Sasaki and Kazumaro Aoki. 2009. Finding preimages in full md5 faster than exhaustive search. In *Advances in Cryptology-EUROCRYPT 2009: 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings 28*, pages 134–152. Springer.

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.

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

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. 2023. Cognitive architectures for language agents.

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

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

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

Lei Wang, Jingsen Zhang, 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.

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

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

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 (Volume 1: Long Papers)*, pages 13484–13508, Toronto, Canada. Association for Computational Linguistics.

Zhilin Wang, Yu Ying Chiu, and Yu Cheung Chiu. 2023f. Humanoid agents: Platform for simulating human-like generative agents. *arXiv preprint arXiv:2310.05418*.

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

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

Lilian Weng. 2023. Llm-powered autonomous agents. 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.

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

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2023b. GPT4Tools: Teaching Large Language Model to Use Tools via Self-instruction. In *Advances in 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.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2023. Expel: Llm agents are experiential learners.

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

Wangchunshu Zhou, Yuchen Eleanor Jiang, Long Li, Jialong Wu, Tiannan Wang, Shi Qiu, Jintian Zhang, Jing Chen, Ruipu Wu, Shuai Wang, Shiding Zhu, Jiyu Chen, Wentao Zhang, Ningyu Zhang, Huajun Chen, Peng Cui, and Mrinmaya Sachan. 2023b. Agents: An open-source framework for autonomous language agents.

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. *arXiv preprint arXiv:2305.17144*.