

ChatDev: Communicative Agents for Software Development

Anonymous ACL submission

Abstract

Software development is a complex task that necessitates cooperation among multiple members with diverse skills. Numerous studies used deep learning to improve specific phases in a waterfall model, such as design, coding, and testing. However, the deep learning model in each phase requires unique designs, leading to technical inconsistencies across various phases, which results in a fragmented and ineffective development process. In this paper, we introduce ChatDev, a chat-powered software development framework in which specialized agents driven by large language models (LLMs) are guided in what to communicate (via *chat chain*) and how to communicate (via *communicative dehallucination*). These agents actively contribute to the design, coding, and testing phases through unified language-based communication, with solutions derived from their multi-turn dialogues. We found their utilization of natural language is advantageous for system design, and communicating in programming language proves helpful in debugging. The paradigm shows how LLM-powered agents engage in autonomous collaboration through linguistic communication, establishing language as a unifying bridge in solving complex tasks.

1 Introduction

Software development is a complex task that necessitates cooperation among multiple members with diverse skills (*e.g.*, architects, programmers, and testers) (Basili, 1989; Sawyer and Guinan, 1998). This entails extensive communication among different roles to understand and analyze requirements through natural language, while also encompassing development and debugging using programming languages (Ernst, 2017; Banker et al., 1998). Numerous studies use deep learning to improve specific phases of the waterfall model in software development, such as design, coding, and testing (Pudlitz et al., 2019; Martín and Abran, 2015;

Gao et al., 2019; Wang et al., 2016). Due to these technical inconsistencies, methods employed in different phases remain isolated until now. Every phase, from data collection and labeling to model training and inference, requires its unique designs, leading to a fragmented and less efficient development process in the field (Freeman et al., 2001; Ernst, 2017; Winkler et al., 2020).

Recently, large language models (LLMs) have led to substantial transformations due to their ability to effortlessly integrate extensive knowledge expressed in language (Brown et al., 2020; Bubeck et al., 2023), combined with their strong capacity for role-playing within designated roles (Park et al., 2023; Hua et al., 2023; Chen et al., 2023b). This advancement eliminates the need for model-specific designs and delivers impressive performance in diverse downstream applications. Furthermore, autonomous agents (Richards, 2023; Zhou et al., 2023a) have gained attention for enhancing the capabilities of LLMs with advanced features such as context-aware memory (Sumers et al., 2023), multi-step planning (Liu et al., 2023), and strategic tool using (Schick et al., 2023). Motivated by the recognized potentials, we are inspired to establish language as a unifying bridge—utilizing multiple LLM-powered agents with specialized roles for cooperative software development through language-based communication across different phases; solutions in different phases are derived from their multi-turn dialogues, whether dealing with text or code.

Nevertheless, due to the tendency of LLM hallucinations (Dhuliawala et al., 2023; Zhang et al., 2023b), the strategy of generating software through communicative agents could lead to the non-trivial challenge of *coding hallucinations*, which involves the generation of source code that is incomplete, unexecutable, or inaccurate, ultimately failing to fulfill the intended requirements (Agnihotri and Chug, 2020). The frequent occurrence of coding hallucin-

83 nation in turn reflects the constrained autonomy of
84 agents in task completion, inevitably demanding
85 additional manual intervention and thereby hindering
86 the immediate usability and reliability of the
87 generated software (Ji et al., 2023).

88 In this paper, we propose ChatDev, a *chat*-
89 powered software-development framework inte-
90 grating multiple "software agents" for active in-
91 volvement in three core phases of the software life-
92 cycle: design, coding, and testing. Technically,
93 ChatDev uses a *chat chain* to divide each phase
94 into smaller subtasks further, enabling agents' multi-turn
95 communications to cooperatively propose and develop solutions (*e.g.*, creative ideas or source code). The chain-style communication
96 guides agents on what to communicate, fostering
97 cooperation and smoothly linking natural- and
98 programming-language subtasks to propel problem-
99 solving. Additionally, to minimize coding hallucina-
100 tions, ChatDev includes an *communicative dehal-*
101 *lucination* mechanism, enabling agents to actively
102 request more specific details before giving direct
103 responses. The communication pattern instructs
104 agents on how to communicate, enabling precise
105 information exchange for effective solution opti-
106 mization while reducing coding hallucinations. We
107 built a comprehensive dataset containing software
108 requirement descriptions and conducted compre-
109 hensive analyses. The results indicate that ChatDev
110 notably improves the quality of software, leading
111 to improved completeness, executability, and better
112 consistency with requirements. Further investiga-
113 tions reveal that natural-language communications
114 contribute to comprehensive system design, while
115 programming-language communications drive soft-
116 ware optimization.

119 2 Related Work

120 Software development is a multifaceted and intricate process that requires the cooperation of mul-
121 tiple experts from various fields (Yilmaz et al.,
122 2012; Acuna et al., 2006; Basili, 1989; Sawyer
123 and Guinan, 1998; Banker et al., 1998; France
124 and Rumpe, 2007), encompassing the require-
125 ment analysis and system design in natural lan-
126 guages (Pudlitz et al., 2019; Martín and Abran,
127 2015; Nahar et al., 2022), along with system de-
128 velopment and debugging in programming lan-
129 guages (Gao et al., 2019; Wang et al., 2016; Wan
130 et al., 2022). Numerous studies employ the wa-
131 terfall model, a particular software development

133 life cycle, to segment the software development
134 process into discrete phases (*e.g.*, design, coding,
135 testing) and apply deep learning techniques to im-
136 prove the effectiveness of certain phases (Winkler
137 et al., 2020; Ezzini et al., 2022; Thaller et al., 2019;
138 Zhao et al., 2021; Nijkamp et al., 2023; Wan et al.,
139 2018; Wang et al., 2021).

140 Trained on vast datasets to comprehend and ma-
141 nicipulate billions of parameters, LLMs have be-
142 come pivotal in natural language processing due
143 to their seamless integration of extensive knowl-
144 edge (Brown et al., 2020; Bubeck et al., 2023;
145 Vaswani et al., 2017; Radford et al.; Touvron et al.,
146 2023; Wei et al., 2022a; Shanahan et al., 2023;
147 Chen et al., 2021; Brants et al., 2007; Chen et al.,
148 2021; Ouyang et al., 2022; Yang et al., 2023a; Qin
149 et al., 2023b; Kaplan et al., 2020). Furthermore,
150 LLMs have demonstrated strong role-playing abili-
151 ties in assigned roles (Li et al., 2023a; Park et al.,
152 2023; Hua et al., 2023; Chan et al., 2023; Zhou
153 et al., 2023b; Chen et al., 2023b,a; Cohen et al.,
154 2023; Li et al., 2023b). Recent progress, particu-
155 larly in the field of autonomous agents (Zhou et al.,
156 2023a; Wang et al., 2023a; Park et al., 2023; Wang
157 et al., 2023e; Richards, 2023; Osika, 2023; Wang
158 et al., 2023d), is largely attributed to the founda-
159 tional advances in LLMs. These agents utilize the
160 robust capabilities of LLMs, displaying remarkable
161 skills in memory (Park et al., 2023; Sumers et al.,
162 2023), planning (Chen et al., 2023b; Liu et al.,
163 2023) and tool use (Schick et al., 2023; Cai et al.,
164 2023; Qin et al., 2023a; Ruan et al., 2023; Yang
165 et al., 2023b), enabling them to reason or operate in
166 complex scenarios (Wei et al., 2022b; Zhao et al.,
167 2023; Zhou et al., 2023a; Ma et al., 2023; Zhang
168 et al., 2023a; Wang et al., 2023b; Ding et al., 2023;
169 Weng, 2023).

170 3 ChatDev

171 We introduce ChatDev, a *chat*-powered software-
172 development framework that integrates multiple
173 "software agents" with various social roles (*e.g.*,
174 requirements analysts, professional programmers
175 and test engineers) collaborating in the core phases
176 of the software life cycle, as visualized in Fig-
177 ure 1. Technically, to facilitate cooperative com-
178 munication, ChatDev introduces *chat chain* to fur-
179 ther break down each phase into smaller and manag-
180 able subtasks, which guides multi-turn communi-
181 cations between different roles to propose and val-
182 idate solutions for each subtask. In addition, to

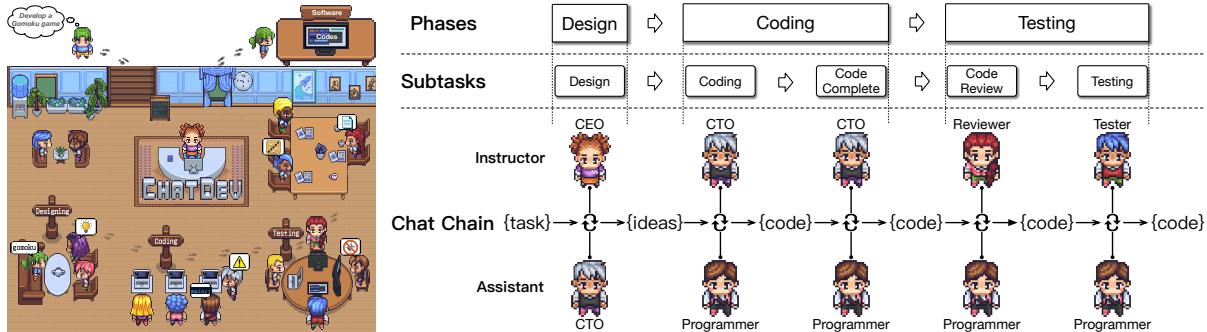


Figure 1: ChatDev, a chat-powered software development framework, brings together LLM agents with various social roles such as requirements analysts, professional programmers and test engineers (left; please zoom in for more details). Upon receiving a preliminary task requirement (*e.g.*, “develop a gomoku game”), the agents engage in multi-turn communication and perform instruction-following along a chat chain, collaborating to execute a series of subtasks autonomously to craft a comprehensive solution (right).

alleviate unexpected hallucinations, a communicative pattern named *communicative dehallucination* is devised, wherein agents request more detailed information before responding directly and then continue the next round of communication based on these details.

3.1 Chat Chain

Although LLMs show a good understanding of natural and programming languages, efficiently transforming textual requirements into functional software in a single step remains a significant challenge. ChatDev thus adopts the core principles of the waterfall model, using a chat chain (\mathcal{C}) with sequential phases (\mathcal{P}), each comprising sequential subtasks (\mathcal{T}). Specifically, ChatDev segments the software development process into three sequential phases: design, coding, and testing. The coding phase is further subdivided into subtasks of code writing and completion, and the testing phase is segmented into code review (static testing) and system testing (dynamic testing), as illustrated in Figure 1. In every subtask, two agents, each with their own specialized roles (*e.g.*, a reviewer skilled at identifying endless loops and a programmer adept in GUI design), perform the functions of an instructor (\mathcal{I}) and an assistant (\mathcal{A}). The instructor agent initiates instructions, instructing (\rightarrow) the discourse toward the completion of the subtask, while the assistant agent adheres to these instructions and responds with (\rightsquigarrow) appropriate solutions. They engage in a multi-turn dialogue (\mathcal{C}), working cooperatively until they achieve consensus, extracting (τ) solutions that can range from the text (*e.g.*, defining a software function point) to code (*e.g.*, creating the initial version of source code), ultimately lead-

ing to the completion of the subtask. The entire task-solving process along the chat chain can be formulated as:

$$\begin{aligned} \mathcal{C} &= \langle \mathcal{P}^1, \mathcal{P}^2, \dots, \mathcal{P}^{|\mathcal{C}|} \rangle \\ \mathcal{P}^i &= \langle \mathcal{T}^1, \mathcal{T}^2, \dots, \mathcal{T}^{|\mathcal{P}^i|} \rangle \\ \mathcal{T}^j &= \tau(\mathcal{C}(\mathcal{I}, \mathcal{A})) \\ \mathcal{C}(\mathcal{I}, \mathcal{A}) &= \langle \mathcal{I} \rightarrow \mathcal{A}, \mathcal{A} \rightsquigarrow \mathcal{I} \rangle_{\circ} \end{aligned} \quad (1)$$

The dual-agent communication design simplifies communications by avoiding complex multi-agent topologies, effectively streamlining the consensus-reaching process (Yin et al., 2023; Chen et al., 2023b). Subsequently, the solutions from previous tasks serve as bridges to the next phase, allowing a smooth transition between subtasks. This approach continues until all subtasks are completed. It’s worth noting that the conceptually simple but empirically powerful chain-style structure guides agents on what to communicate, fostering cooperation and smoothly linking natural- and programming-language subtasks. It also offers a transparent view of the entire software development process, allowing for the examination of intermediate solutions and assisting in identifying possible problems.

Agentization To enhance the quality and reduce human intervention, ChatDev implements prompt engineering that only takes place at the start of each subtask round. As soon as the communication phase begins, the instructor and the assistant will communicate with each other in an automated loop, continuing this exchange until the task concludes. However, simply exchanging responses cannot achieve effective multi-round task-oriented communication, since it inevitably faces significant

challenges including role flipping, instruction repeating, and fake replies. As a result, there is a failure to advance the progression of productive communications and hinders the achievement of meaningful solutions. ChatDev thus employs inception prompting mechanism (Li et al., 2023a) for initiating, sustaining, and concluding agents' communication to guarantee a robust and efficient workflow. This mechanism is composed of the instructor system prompt P_I and the assistant system prompt P_A . The system prompts for both roles are mostly symmetrical, covering the overview and objectives of the current subtask, specialized roles, accessible external tools, communication protocols, termination conditions, and constraints or requirements to avoid undesirable behaviors. Then, an instructor \mathcal{I} and an assistant \mathcal{A} are instantiated by hypnotizing LLMs via P_I and P_A :

$$\mathcal{I} = \rho(LLM, P_I), \quad \mathcal{A} = \rho(LLM, P_A) \quad (2)$$

where ρ is the role customization operation, implemented via system message assignment.

Memory Note that the limited context length of common LLMs typically restricts the ability to maintain a complete communication history among all agents and phases. To tackle this issue, based on the nature of the chat chain, we accordingly segment the agents' context memories based on their sequential phases, resulting in two functionally distinct types of memory: *short-term memory* and *long-term memory*. Short-term memory is utilized to sustain the continuity of the dialogue within a single phase, while long-term memory is leveraged to preserve contextual awareness across different phases.

Formally, short-term memory records an agent's current phase utterances, aiding context-aware decision-making. At the time t during phase \mathcal{P}^i , we use \mathcal{I}_t^i to represent the instructor's instruction and \mathcal{A}_t^i for the assistant's response. The short-term memory \mathcal{M} collects utterances up to time t as:

$$\mathcal{M}_t^i = \langle (\mathcal{I}_1^i, \mathcal{A}_1^i), (\mathcal{I}_2^i, \mathcal{A}_2^i), \dots, (\mathcal{I}_t^i, \mathcal{A}_t^i) \rangle \quad (3)$$

In the next time step $t + 1$, the instructor utilizes the current memory to generate a new instruction \mathcal{I}_{t+1}^i , which is then conveyed to the assistant to produce a new response \mathcal{A}_{t+1}^i . The short-term memory iteratively updates until the number of communications reaches the upper limit $|\mathcal{M}^i|$:

$$\begin{aligned} \mathcal{I}_{t+1}^i &= \mathcal{I}(\mathcal{M}_t^i), \quad \mathcal{A}_{t+1}^i = \mathcal{A}(\mathcal{M}_t^i, \mathcal{I}_{t+1}^i) \\ \mathcal{M}_{t+1}^i &= \mathcal{M}_t^i \cup (\mathcal{I}_{t+1}^i, \mathcal{A}_{t+1}^i) \end{aligned} \quad (4)$$

To perceive dialogues through previous phases, the chat chain only transmits the solutions from previous phases as long-term memories $\tilde{\mathcal{M}}$, integrating them at the start of the next phase and enabling the cross-phase transmission of long dialogues:

$$\mathcal{I}_1^{i+1} = \tilde{\mathcal{M}}^i \cup \mathcal{P}_{\mathcal{I}}^{i+1}, \quad \tilde{\mathcal{M}}^i = \bigcup_{j=1}^i \tau(\mathcal{M}_{|\mathcal{M}^j|}^j) \quad (5)$$

where \mathcal{P} symbolizes a predetermined prompt that appears exclusively at the start of each phase.

By sharing only the solutions of each subtask rather than the entire communication history, ChatDev minimizes the risk of being overwhelmed by too much information, enhancing concentration on each task and encouraging more targeted cooperation, while simultaneously facilitating cross-phase context continuity.

3.2 Communicative Dehallucination

LLM hallucinations manifest when models generate outputs that are nonsensical, factually incorrect, or inaccurate (Dhuliawala et al., 2023; Zhang et al., 2023b). This issue is particularly concerning in software development, where programming languages demand precise syntax—the absence of even a single line can lead to system failure. We have observed that LLMs often produce *coding hallucinations*, which encompass potential issues like incomplete implementations, unexecutable code, and inconsistencies that don't meet requirements. Coding hallucinations frequently appear when the assistant struggles to precisely follow instructions, often due to the vagueness and generality of certain instructions that require multiple adjustments, making it challenging for agents to achieve full compliance. Inspired by this, we introduce *communicative dehallucination*, which encourages the assistant to actively seek more detailed suggestions from the instructor before delivering a formal response.

Specifically, a vanilla communication pattern between the assistant and the instructor follows a straightforward instruction-response format:

$$\langle \mathcal{I} \rightarrow \mathcal{A}, \quad \mathcal{A} \rightsquigarrow \mathcal{I} \rangle_{\circ} \quad (6)$$

In contrast, our communicative dehallucination mechanism features a deliberate "role reversal", where the assistant takes on an instructor-like role, proactively seeking more specific information (e.g., the precise name of an external dependency and

343 its related class) before delivering a conclusive re-
344 sponse. After the instructor provides a specific
345 modification suggestion, the assistant proceeds to
346 perform precise optimization:

347 $\langle \mathcal{I} \rightarrow \mathcal{A}, \langle \mathcal{A} \rightarrow \mathcal{I}, \mathcal{I} \sim \mathcal{A} \rangle_{\odot}, \mathcal{A} \sim \mathcal{I} \rangle_{\odot}$ (7)

348 Since this mechanism tackles one concrete issue at
349 a time, it requires multiple rounds of communica-
350 tion to optimize various potential problems. The
351 communication pattern instructs agents on how to
352 communicate, enabling finer-grained information
353 exchange for effective solution optimization, which
354 practically aids in reducing coding hallucinations.

355 4 Evaluation

356 **Baselines** We chose some representative LLM-
357 based software development methods as our base-
358 lines. GPT-Engineer (Osika, 2023) is a fundamen-
359 tal single-agent approach in LLM-driven software
360 agents with a precise understanding of task require-
361 ments and the application of one-step reasoning,
362 which highlights its efficiency in generating de-
363 tailed software solutions at the repository level.
364 MetaGPT (Hong et al., 2023) is an advanced frame-
365 work that allocates specific roles to various LLM-
366 driven software agents and incorporates standard-
367 ized operating procedures to enable multi-agent
368 participation. In each step agents with specific roles
369 generate solutions by adhering to static instructions
370 predefined by human experts.

371 **Datasets** Note that, as of now, there isn't a pub-
372 licly accessible dataset containing textual descrip-
373 tions of software requirements in the context of
374 agent-driven software development. To this end,
375 we are actively working towards developing a com-
376 prehensive dataset for software requirement de-
377 scriptions, which we refer to as SRDD (Software
378 Requirement Description Dataset). Drawing on
379 previous work (Li et al., 2023a), we utilize existing
380 software descriptions as initial examples, which
381 are then further developed through a process that
382 combines LLM-based automatic generation with
383 post-processing refinement guided by humans. As
384 a result, this dataset includes important software
385 categories from popular platforms such as Ubuntu,
386 Google Play, Microsoft Store, and Apple Store. It
387 comprises 1,200 software task prompts that have
388 been carefully categorized into 5 main areas: Edu-
389 cation, Work, Life, Game, and Creation. All these
390 areas are further divided into 40 subcategories, and
391 each subcategory contains 30 unique task prompts.

Metrics Evaluating software is also a challenging
task, especially when trying to assess it on a holistic
level. Under the current limitation of scarce bench-
mark resources, traditional function-oriented code
generation metrics (e.g., pass@k), cannot seam-
lessly transfer to a comprehensive evaluation of
entire software systems. The main reason for this
is that it is often impractical to develop manual or
automated test cases for various types of software,
especially those involving complex interfaces, fre-
quent user interactions, or non-deterministic feed-
back. As an initial strategy, we apply three funda-
mental and objective dimensions that reflect differ-
ent aspects of coding hallucinations to evaluate the
agent-generated software, and then integrate them
to facilitate 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 "placeholder" code snippets. A higher score indicates a higher probability of automated completion.
- *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* measures how closely the generated software code aligns with the original requirement description, quantified as the cosine distance between the semantic embeddings of the textual requirements and the generated software code¹. A higher score indicates a greater degree of consistency with the requirements.
- *Quality* is a comprehensive metric that integrates various factors to assess the overall quality of software, quantified by multiplying completeness, executability, and consistency. A higher quality score suggests a higher overall satisfaction with the software generated, implying a lower need for further manual intervention.

Implementation Details We divided software development into 5 subtasks within 3 phases, assigning specific roles like CEO, CTO, programmer, reviewer, and tester. A subtask would terminate and get a conclusion either after two unchanged

¹Comments should be excluded from the code to avoid potential information leakage during evaluations.

| Method | Paradigm | Completeness | Executability | Consistency | Quality |
|--------------|----------|----------------------------|----------------------------|----------------------------|----------------------------|
| GPT-Engineer | 😊 | <u>0.5022</u> [†] | 0.3583 [†] | <u>0.7887</u> [†] | 0.1419 [†] |
| MetaGPT | 😊😊 | 0.4834 [†] | <u>0.4145</u> [†] | 0.7601 [†] | <u>0.1523</u> [†] |
| ChatDev | 😊😊 | 0.5600 | 0.8800 | 0.8021 | 0.3953 |

Table 1: Overall performance of the LLM-powered software development methods, encompassing both single-agent (😊) and multi-agent (😊😊) paradigms. Performance metrics are averaged for all tasks. The top scores are in bold, with second-highest underlined. † indicates significant statistical differences ($p \leq 0.05$) between a baseline and ours.

| Method | Evaluator | Baseline Wins | ChatDev Wins | Draw |
|--------------|-----------|---------------|--------------|--------|
| GPT-Engineer | GPT-4 | 22.50% | 77.08% | 00.42% |
| | Human | 09.18% | 90.16% | 00.66% |
| MetaGPT | GPT-4 | 37.50% | 57.08% | 05.42% |
| | Human | 07.92% | 88.00% | 04.08% |

Table 2: Pairwise evaluation results.

439 code modifications or after 10 rounds of communication. During the code completion, review, and
440 testing, a communicative dehallucination is activated.
441 For ease of identifying solutions, the assistant begins responses with "<SOLUTION>" when
442 a consensus is reached. We used ChatGPT-3.5 with
443 a temperature of 0.2 and integrated Python-3.11.4
444 for feedback. All baselines in the evaluation share
445 the same hyperparameters and settings for fairness.
446

448 4.1 Overall Performance

449 As illustrated in Table 1, ChatDev outperforms
450 all baseline methods across all metrics, showing
451 a considerable margin of improvement. Firstly,
452 the improvement of ChatDev and MetaGPT over
453 GPT-Engineer demonstrates that complex tasks are
454 difficult to solve in a single-step solution. There-
455 fore, explicitly decomposing the difficult problem
456 into several smaller, more manageable subtasks en-
457 hances the effectiveness of task completion. Addi-
458 tionally, in comparison to MetaGPT, ChatDev sig-
459 nificantly raises the *Quality* from 0.1523 to 0.3953.
460 This advancement is largely attributed to the agents
461 employing a cooperative communication method,
462 which involves autonomously proposing and con-
463 tinuously refining source code through a blend of
464 natural and programming languages, as opposed
465 to merely delivering responses based on human-
466 predefined instructions. The communicative agents
467 guide each subtask towards integrated and auto-
468 mated solutions, efficiently overcoming the restric-
469 tions typically linked to manually established op-
470 timization rules, and offering a more versatile and
471 adaptable framework for problem-solving.

472 To further understand user preferences in practi-

| Method | Duration (s) | #Tokens | #Files | #Lines |
|--------------|--------------|-------------|--------|----------|
| GPT-Engineer | 15.6000 | 7,182.5333 | 3.9475 | 70.2041 |
| MetaGPT | 154.0000 | 29,278.6510 | 4.4233 | 153.3000 |
| ChatDev | 148.2148 | 22,949.4450 | 4.3900 | 144.3450 |

Table 3: Software statistics include Duration (time consumed), #Tokens (number of tokens used), #Files (number of code files generated), and #Lines (total lines of code across all files) in the software generation process.

473 cal settings, we use the setting adopted by Li et al.
474 (2023a), where agent-generated solutions are com-
475 pared in pairs by both human participants and the
476 prevalent GPT-4 model to identify the preferred
477 one.² Table 2 shows ChatDev consistently outper-
478 forming other baselines, with higher average win
479 rates in both GPT-4 and human evaluations.

480 Furthermore, the software statistics presented
481 in Table 3 indicates that the multi-agent paradigm,
482 despite being slower and consuming more tokens
483 than the single-agent method, yields a greater num-
484 ber of code files and a larger codebase, which may
485 enhance the software's functionality and integrity.
486 Analyzing the dialogues of agents suggests that
487 the multi-agent communication method often leads
488 agents to autonomously offer functional enhance-
489 ments (e.g., GUI creation or increasing game diffi-
490 culty), thereby potentially resulting in the incorpo-
491 ration of beneficial features that were not explicitly
492 specified in requirements. Taking all these factors
493 together, we posit that the fundamental character-
494 istics of multi-agent software development take
495 on greater significance, surpassing short-term con-
496 cerns like time and economic costs in the current
497 landscape.

498 4.2 Ablation Study

499 This section examines key components or mech-
500 anisms within our multi-agent cooperation frame-
501 work by removing particular phases in the chat

²For fairness, GPT-4's evaluation mitigated possible pos-
502itional bias (Wang et al., 2023c), and human experts inde-
503pendently assessed the task solutions, randomized to prevent
504order bias.

| Variant | Completeness | Executability | Consistency | Quality |
|-------------------|---------------|---------------|---------------|---------------|
| ChatDev | 0.5600 | 0.8800 | 0.8021 | 0.3953 |
| \leq Coding | 0.4100 | 0.7700 | 0.7958 | 0.2512 |
| \leq Complete | 0.6250 | 0.7400 | 0.7978 | 0.3690 |
| \leq Review | <u>0.5750</u> | <u>0.8100</u> | <u>0.7980</u> | <u>0.3717</u> |
| \leq Testing | 0.5600 | 0.8800 | 0.8021 | 0.3953 |
| \setminus CDH | 0.4700 | 0.8400 | 0.7983 | 0.3094 |
| \setminus Roles | 0.5400 | 0.5800 | 0.7385 | 0.2212 |

Table 4: Ablation study on main components or mechanisms. $\leq x$ denotes halting the Chat Chain after the completion of the x phrase, and \setminus denotes the removing operation. CDH denotes the communicative dehallucination mechanism.

chain, communicative dehallucination, or the roles assigned to all agents in their system prompts. Figure 4 shows that the code complete phase enhances *Completeness*, with testing critical for *Executability*. *Quality* steadily rises with each step, suggesting that software development optimization is progressively attained through multi-phase communications among intelligent agents. Meanwhile, eliminating communicative dehallucination results in a decrease across all metrics, indicating its effectiveness in addressing coding hallucinations. Most interestingly, the most substantial impact on performance occurs when the roles of all agents are removed from their system prompts. Detailed dialogue analysis shows that assigning a "prefer GUI design" role to a programmer results in generated source code with relevant GUI implementations; in the absence of such role indications, it defaults to implement unfriendly command-line-only programs only. Likewise, assigning roles such as a "careful reviewer for bug detection" enhances the chances of discovering code vulnerabilities; without such roles, feedback tends to be high-level, leading to limited adjustments by the programmer. This finding underscores the importance of assigning roles in eliciting responses from LLMs, underscoring the significant influence of multi-agent cooperation on software quality.

4.3 Communication Analysis

Our agent-driven software development paradigm promotes cooperative agents through effective communication for automated solution optimization. Phases in the chat chain have varying levels of engagement in natural and programming languages. We now analyze the content of their communications to understand linguistic effects.

Figure 2 depicts a communication breakdown, with natural language at 57.20%. In the natural-language phase (*i.e.*, design), natural language com-

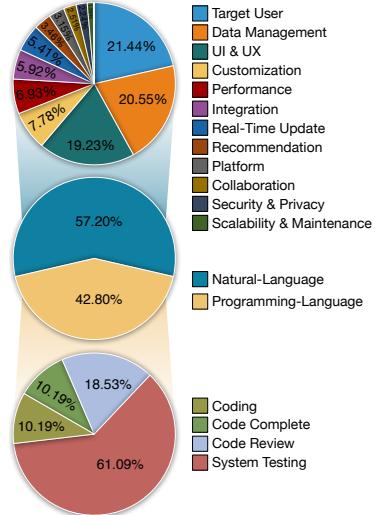


Figure 2: The utterance distribution of agent communications throughout the entire development process.

munication plays a crucial role in the thorough design of the system, with agents autonomously discussing and designing aspects like target user, data management, and user interface. Post-design phases show a balanced mix of coding, code completion, and testing activities, with most communication occurring during code reviews. This trend is due to agents' self-reviews and code fixes consistently propelling software development; otherwise, progress halts when successive updates don't show significant changes, leading to a natural decrease in code review communications.

We explore the properties of static debugging dynamics in code reviews resulting from communication between reviewers and programmers, as depicted in Figure 3. The data uncovers that during the review phase, reviewers may spot different issues through language interactions. The programmer's intervention can transform certain issues into different ones or a state where no further suggestions are needed; the increasing proportion of the latter indicates successful software optimization. Particularly, the "Method Not Implemented" issue is most common in communication between reviewers and programmers during code reviews, accounting for 34.85% of discussions. This problem usually arises from unclear text requirements and the use of "placeholder" tags in Python code, necessitating additional manual adjustments. Furthermore, the "Module Not Imported" issue often arises due to code generation omitting crucial details. Apart from common problems, reviewers often focus on enhancing code robustness by identifying rare exceptions, unused classes, or potential infinite loops.

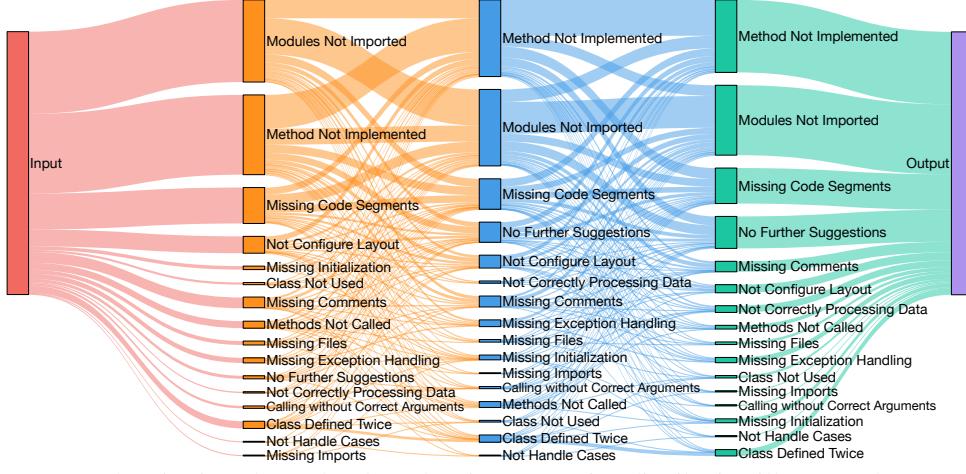


Figure 3: In a 3-round reviewing phase, the chart showing suggestion distribution illustrates the recommendations made by a reviewer agent, where each sector represents a different category of suggestion.

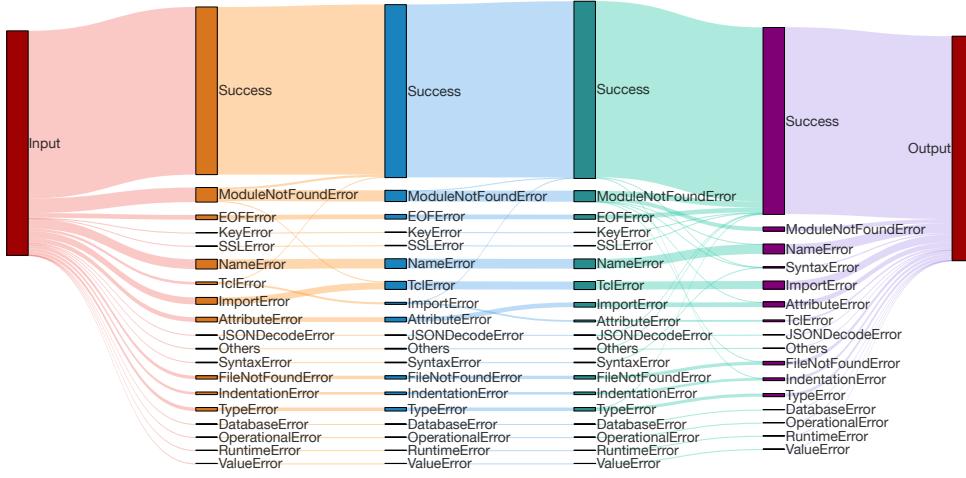


Figure 4: During the testing phase, the software state transition diagram depicts the evolution of iterations, with each colored column in the diagram symbolizing a dialogue round.

Likewise, we analyze the tester-programmer communication during the testing phase, illustrating the dynamic debugging dynamics in their multi-turn interactions with compiler feedback, as depicted in Figure 4. The likelihood of successful compilation at each step is generally higher than encountering errors, with most errors persisting and a lower probability of transforming into different errors. The most frequent error is "ModuleNotFound" (45.76%), followed by "NameError" and "ImportError" (each at 15.25%). The observation highlights the model's tendency to overlook basic elements like an "import" statement, underscoring its difficulty in managing intricate details during code generation. Besides, the tester also detects rarer errors like improperly initialized GUIs, incorrect method calls, missing file dependencies, and unused modules. The communicative dehallucination mechanism effectively resolves certain errors, frequently resulting in "compilation success" after code changes.

There's a significantly low chance of returning to an error state from a successful compilation. Over time, the multi-turn communication process statistically shows a consistent decrease in errors, steadily moving towards successful software execution.

5 Conclusion

We have introduced ChatDev, an innovative autonomous agent dialogue framework for software development that utilizes multiple LLM-powered agents to integrate fragmented phases of the waterfall model into a cohesive communication system. It features *chat chain* organizing communication targets and *dehallucination* for resolving coding hallucinations. The results demonstrate its superiority and highlight the benefits of multi-turn communications in software optimization. We aim for the insights to advance LLM agents towards increased autonomy and illuminate the profound effects of "language" and its empowering role across an even broader spectrum of applications.

6 Limitations

Our study has explored the capabilities of cooperative autonomous agents in software development, yet both researchers and practitioners must be mindful of certain limitations or risks when using these insights to develop new techniques or applications.

Firstly, from the perspective of software capabilities, the ability of autonomous agents to produce software might be overestimated. From our observations, even though the approach effectively enhances the quality of software development, we find that autonomous agents tend to implement the simplest logic during the software production process, *i.e.*, relatively low information density produced. Without necessary and clear requirements, it is difficult for agents to autonomously identify the actual ideas behind the task descriptions. For instance, in the development of a Snake game, if the task guidelines are not precisely outlined, agents tend to represent the snake with a basic rectangular shape and use a simple circle to symbolize the target food. Similarly, when building an information management system, agents tend to embed the information to be queried into the code in a basic key-value format, rather than opting for a more flexible, external database solution. Hence, we recommend clearly defining detailed software requirements in the task, such as specifying if an interface is necessary, if there's a need for automatic generation of game character assets, or if an external database is required. Considering the current capabilities of agents, ensuring the fulfillment of highly detailed requirements is not always guaranteed, emphasizing the need to find a balance between specificity and practical feasibility in the requirements. To sum up, this research is still in its early phases of development, and the relevant technologies are not yet easily adaptable to complex real-world environments. Consequently, at this phase, its application is more appropriate for developing prototype systems rather than full-fledged real-world software systems.

Secondly, in terms of evaluation, this area diverges from the traditional code generation approach, which is limited to creating function-level code. Software development involves sophisticated communication between natural language processing and programming language skills. Despite the progress made by LLMs, they still encounter challenges in complex areas such as requirement analysis, architectural design, code review, and software

testing. Due to the intricate nature of software, automating the evaluation of general-purpose software is exceptionally challenging. Recent efforts aimed to employ *Human Revision Cost*³ (Hong et al., 2023) across seven software tasks; however, manual verification proved to be excessively labor-intensive or even impractical when dealing with the large-scale dataset. This paper instead focuses primarily on objective and crucial dimensions: completeness, executability, consistency, and a comprehensive quality. However, there is a need for future research to delve into additional dimensions, including functionalities, robustness, safety, user-friendliness, and more. Crafting a completely fair and objective evaluation standard is a significant challenge. Therefore, for the foreseeable future, software evaluation might require customization for specific scenarios and the integration of human feedback, which can more accurately steer the development of software processes in a way that better meets human requirements.

Thirdly, in comparison to the single-agent approach, the iterative exchange between multiple agents naturally requires more tokens and time, leading to increased computational demand for the underlying models, which could have an environmental impact. Consequently, future research should concentrate on developing methods that allow agents to achieve equivalent or stronger capabilities through fewer interactions.

However, we hold the belief that these potential limitations serve as inspiration for future directions and can be effectively mitigated by drawing a broader technically proficient audience. We expect that these findings have the potential to provide valuable insights into enhancing cooperative task-solving in the ever-evolving dynamics of LLM-powered agents.

References

- Silvia T Acuna, Natalia Juristo, and Ana M Moreno. 2006. [Emphasizing Human Capabilities in Software Development](#). In *IEEE Software*, volume 23, pages 94–101.

- Mansi Agnihotri and Anuradha Chug. 2020. [A Systematic Literature Survey of Software Metrics, Code Smells and Refactoring Techniques](#). In *booktitle of*

³Human Revision Cost is quantified by the number of rounds of revision needed to ensure the smooth running of the code, this indicates the frequency of human interventions, such as debugging or importing packages.

| | | |
|-----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------|
| 714 715 | Information Processing Systems, volume 16, pages 915–934. | 770 771 772 773 774 775 |
| 716 717 718 719 720 | Rajiv D Bunker, Gordon B Davis, and Sandra A Slaugher. 1998. Software Development Practices, Software Complexity, and Software Maintenance Performance: A Field Study. In <i>Management science</i> , volume 44, pages 433–450. | |
| 721 722 723 724 | Victor R Basili. 1989. Software Development: A Paradigm for The Future. In <i>Proceedings of the Annual International Computer Software and Applications Conference</i> , pages 471–485. IEEE. | |
| 725 726 727 728 729 730 731 | Thorsten Brants, Ashok C Popat, Peng Xu, Franz J Och, and Jeffrey Dean. 2007. Large Language Models in Machine Translation. In <i>Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)</i> , pages 858–867. | |
| 732 733 734 735 736 737 738 739 740 741 742 743 744 | 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 <i>Advances in Neural Information Processing Systems (NeurIPS)</i> , volume 33, pages 1877–1901. | |
| 745 746 747 748 749 750 751 | Sébastien Bubeck, Varun Chandrasekaran, Ronen Elidan, 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. In <i>arXiv preprint arXiv:2303.12712</i> . | |
| 752 753 754 | Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2023. Large Language Models as Tool Makers. In <i>arXiv preprint arXiv:2305.17126</i> . | |
| 755 756 757 758 759 | 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. In <i>arXiv preprint arXiv:2308.07201</i> . | |
| 760 761 762 763 | Dake Chen, Hanbin Wang, Yunhao Huo, Yuzhao Li, and Haoyang Zhang. 2023a. GameGPT: Multi-agent Collaborative Framework for Game Development. In <i>arXiv preprint arXiv:2310.08067</i> . | |
| 764 765 766 767 768 769 | 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 <i>arXiv preprint arXiv:2107.03374</i> . | |
| | 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. In <i>arXiv preprint arXiv:2308.10848</i> . | |
| | Roi Cohen, May Hamri, Mor Geva, and Amir Globerson. 2023. LM vs LM: Detecting Factual Errors via Cross Examination. In <i>ArXiv</i> , volume abs/2305.13281. | |
| | Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. 2023. Chain-of-Verification Reduces Hallucination in Large Language Models. In <i>arXiv preprint arXiv:2309.11495</i> . | |
| | Shiying Ding, Xinyi Chen, Yan Fang, Wenrui Liu, Yiwu Qiu, and Chunlei Chai. 2023. DesignGPT: Multi-Agent Collaboration in Design. In <i>arXiv preprint arXiv:2311.11591</i> . | |
| | Michael D. Ernst. 2017. Natural Language is a Programming Language: Applying Natural Language Processing to Software Development. In <i>Advances in Programming Languages (SNAPL)</i> , volume 71, pages 4:1–4:14. | |
| | Saad Ezzini, Sallam Abualhaija, Chetan Arora, and Mehrdad Sabetzadeh. 2022. Automated Handling of Anaphoric Ambiguity in Requirements: A Multi-solution Study. In <i>International Conference on Software Engineering (ICSE)</i> , pages 187–199. | |
| | Robert France and Bernhard Rumpe. 2007. Model-driven Development of Complex Software: A Research Roadmap. In <i>Future of Software Engineering (FOSE)</i> , pages 37–54. | |
| | Peter Freeman, Donald J. Bagert, Hossein Saiedian, Mary Shaw, Robert Dupuis, and J. Barrie Thompson. 2001. Software Engineering Body of Knowledge (SWEBOK). In <i>Proceedings of the International Conference on Software Engineering (ICSE)</i> , pages 693–696. | |
| | Ryan J Gallagher, Kyle Reing, David Kale, and Greg Ver Steeg. 2017. Anchored Correlation Explanation: Topic Modeling with Minimal Domain Knowledge. <i>Transactions of the Association for Computational Linguistics</i> , 5:529–542. | |
| | Sa Gao, Chunyang Chen, Zhenchang Xing, Yukun Ma, Wen Song, and Shang-Wei Lin. 2019. A Neural Model for Method Name Generation from Functional Description. In <i>26th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)</i> , pages 411–421. | |
| | 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. In <i>arXiv preprint arXiv:2308.00352</i> . | |

| | | |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 827 | 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 <i>arXiv preprint arXiv:2311.17227</i> . | 883 |
| 828 | | 884 |
| 829 | | 885 |
| 830 | | 886 |
| 831 | | |
| 832 | | |
| 833 | Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of Hallucination in Natural Language Generation. In <i>ACM Computing Surveys</i> , volume 55, pages 1–38. | 889 |
| 834 | | 890 |
| 835 | | 891 |
| 836 | | 892 |
| 837 | | 893 |
| 838 | 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 <i>arXiv preprint arXiv:2001.08361</i> . | 894 |
| 839 | | 895 |
| 840 | | 896 |
| 841 | | 897 |
| 842 | | |
| 843 | Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbulin, and Bernard Ghanem. 2023a. CAMEL: Communicative Agents for "Mind" Exploration of Large Scale Language Model Society. In <i>Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS)</i> . | 898 |
| 844 | | 899 |
| 845 | | 900 |
| 846 | | 901 |
| 847 | | 902 |
| 848 | | 903 |
| 849 | 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 <i>arXiv preprint arXiv:2310.06500</i> . | 904 |
| 850 | | 905 |
| 851 | | 906 |
| 852 | | 907 |
| 853 | | 908 |
| 854 | 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 <i>arXiv preprint arXiv:2308.05960</i> . | 909 |
| 855 | | 910 |
| 856 | | 911 |
| 857 | | 912 |
| 858 | | 913 |
| 859 | | |
| 860 | | |
| 861 | Kaixin Ma, Hongming Zhang, Hongwei Wang, Xiaoman Pan, and Dong Yu. 2023. LASER: LLM Agent with State-Space Exploration for Web Navigation. In <i>arXiv preprint arXiv:2309.08172</i> . | 914 |
| 862 | | 915 |
| 863 | | 916 |
| 864 | | 917 |
| 865 | Cuaughtémoc López Martín and Alain Abran. 2015. Neural networks for predicting the duration of new software projects. In <i>J. Syst. Softw.</i> , volume 101, pages 127–135. | 918 |
| 866 | | 919 |
| 867 | | |
| 868 | | |
| 869 | Stefano Mezza, Alessandra Cervone, Evgeny Stepanov, Giuliano Tortoreto, and Giuseppe Riccardi. 2018. ISO-Standard Domain-Independent Dialogue Act Tagging for Conversational Agents. In <i>Proceedings of the 27th International Conference on Computational Linguistics</i> , pages 3539–3551. | 920 |
| 870 | | 921 |
| 871 | | 922 |
| 872 | | 923 |
| 873 | | |
| 874 | | |
| 875 | Nadia Nahar, Shurui Zhou, Grace A. Lewis, and Christian Kästner. 2022. Collaboration Challenges in Building ML-Enabled Systems: Communication, Documentation, Engineering, and Process. In <i>IEEE/ACM International Conference on Software Engineering (ICSE)</i> , pages 413–425. | 924 |
| 876 | | 925 |
| 877 | | |
| 878 | | |
| 879 | | |
| 880 | | |
| 881 | Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming | 926 |
| 882 | | 927 |
| 883 | | 928 |
| 884 | | 929 |
| 885 | | 930 |
| 886 | | 931 |
| 887 | Xiong. 2023. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. In <i>The International Conference on Learning Representations (ICLR)</i> . | 932 |
| 888 | | 933 |
| 889 | Anton Osika. 2023. GPT-Engineer. In https://github.com/antonosika/gpt-engineer . | 934 |
| 890 | | |
| 891 | | |
| 892 | | |
| 893 | | |
| 894 | 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. In <i>arXiv preprint arXiv:2203.02155</i> . | 935 |
| 895 | | 936 |
| 896 | | 937 |
| 897 | | |
| 898 | 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 <i>Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)</i> . | 938 |
| 899 | | 939 |
| 900 | | 940 |
| 901 | | 941 |
| 902 | | 942 |
| 903 | | 943 |
| 904 | Florian Pudlitz, Florian Brokhausen, and Andreas Vogelsang. 2019. Extraction of System States from Natural Language Requirements. In <i>IEEE International Requirements Engineering Conference (RE)</i> , pages 211–222. | 944 |
| 905 | | 945 |
| 906 | | 946 |
| 907 | | 947 |
| 908 | | 948 |
| 909 | 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. In <i>arXiv preprint arXiv:2307.16789</i> . | 949 |
| 910 | | 950 |
| 911 | | 951 |
| 912 | | 952 |
| 913 | | 953 |
| 914 | 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. In <i>arXiv preprint arXiv:2306.17563</i> . | 954 |
| 915 | | 955 |
| 916 | | 956 |
| 917 | | 957 |
| 918 | | 958 |
| 919 | | 959 |
| 920 | Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language Models are Unsupervised Multitask Learners. In <i>OpenAI Blog</i> , volume 1, page 9. | 960 |
| 921 | | 961 |
| 922 | | 962 |
| 923 | | 963 |
| 924 | Toran Bruce Richards. 2023. AutoGPT. In https://github.com/Significant-Gravitas/AutoGPT . | 964 |
| 925 | | 965 |
| 926 | 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. In <i>arXiv preprint arXiv:2308.03427</i> . | 966 |
| 927 | | 967 |
| 928 | | 968 |
| 929 | | 969 |
| 930 | | 970 |
| 931 | | 971 |
| 932 | Steve Sawyer and Patricia J. Guinan. 1998. Software development: Processes and Performance. In <i>IBM Systems Journal</i> , volume 37, pages 552–569. | 972 |
| 933 | | 973 |
| 934 | | 974 |
| 935 | Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. ToolFormer: | 975 |
| 936 | | 976 |
| 937 | | 977 |

| | | | | |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|--|--|
| 938 | Language Models Can Teach Themselves to Use Tools. | In <i>arXiv preprint arXiv:2302.04761</i> . | | |
| 939 | | | | |
| 940 | Murray Shanahan, Kyle McDonell, and Laria Reynolds. | | | |
| 941 | 2023. Role Play with Large Language Models. | In <i>Nature</i> , volume 623, pages 493–498. | | |
| 942 | | | | |
| 943 | Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, | | | |
| 944 | and Thomas L. Griffiths. 2023. Cognitive Architectures for Language Agents. | In <i>arXiv preprint arXiv:2309.02427</i> . | | |
| 945 | | | | |
| 946 | Hannes Thaller, Lukas Linsbauer, and Alexander Egyed. | | | |
| 947 | 2019. Feature Maps: A Comprehensible Software Representation for Design Pattern Detection. | In <i>IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)</i> , pages 207–217. | | |
| 948 | | | | |
| 949 | 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 <i>arXiv preprint arXiv:2302.13971</i> . | | |
| 950 | | | | |
| 951 | Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All You Need. | In <i>Advances in Neural Information Processing Systems (NeurIPS)</i> , volume 30. | | |
| 952 | | | | |
| 953 | Chengcheng Wan, Shicheng Liu, Sophie Xie, Yifan Liu, Henry Hoffmann, Michael Maire, and Shan Lu. | | | |
| 954 | 2022. Automated Testing of Software that Uses Machine Learning APIs. | In <i>IEEE/ACM International Conference on Software Engineering (ICSE)</i> , pages 212–224. | | |
| 955 | | | | |
| 956 | Yao Wan, Zhou Zhao, Min Yang, Guandong Xu, Haochao Ying, Jian Wu, and Philip S. Yu. 2018. Improving Automatic Source Code Summarization via Deep Reinforcement Learning. | In <i>Proceedings of the ACM/IEEE International Conference on Automated Software Engineering (ASE)</i> , pages 397–407. | | |
| 957 | | | | |
| 958 | 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 <i>arXiv preprint arXiv:2305.16291</i> . | | |
| 959 | | | | |
| 960 | 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 <i>arXiv preprint arXiv:2306.02552</i> . | | |
| 961 | | | | |
| 962 | 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 <i>arXiv preprint arXiv:2305.17926</i> . | | |
| 963 | | | | |
| 964 | Song Wang, Taiyue Liu, and Lin Tan. 2016. Automatically Learning Semantic Features for Defect Prediction. | In <i>Proceedings of the International Conference on Software Engineering (ICSE)</i> , pages 297–308. | | |
| 965 | | | | |
| 966 | Song Wang, Nishtha Shrestha, Abarna Kucherl Subramanian, Junjie Wang, Moshi Wei, and Nachiappan Nagappan. 2021. Automatic Unit Test Generation for Machine Learning Libraries: How Far Are We? | In <i>IEEE/ACM International Conference on Software Engineering (ICSE)</i> , pages 1548–1560. | | |
| 967 | | | | |
| 968 | 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. | In <i>arXiv preprint arXiv:2310.16427</i> . | | |
| 969 | | | | |
| 970 | Zhilin Wang, Yu Ying Chiu, and Yu Cheung Chiu. | | | |
| 971 | 2023e. Humanoid Agents: Platform for Simulating Human-like Generative Agents. | In <i>arXiv preprint arXiv:2310.05418</i> . | | |
| 972 | | | | |
| 973 | Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. | | | |
| 974 | 2022a. Emergent Abilities of Large Language Models. | In <i>arXiv preprint arXiv:2206.07682</i> . | | |
| 975 | | | | |
| 976 | 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. | In <i>Advances in Neural Information Processing Systems (NeurIPS)</i> , volume 35, pages 24824–24837. | | |
| 977 | | | | |
| 978 | Lilian Weng. 2023. LLM-powered Autonomous Agents. | In <i>lilianweng.github.io</i> . | | |
| 979 | | | | |
| 980 | Jonas Winkler, Jannis Grönberg, and Andreas Vogelsang. 2020. Predicting How to Test Requirements: An Automated Approach. | In <i>Software Engineering</i> , volume P-300 of <i>LNI</i> , pages 141–142. | | |
| 981 | | | | |
| 982 | Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2023a. Large Language Models as Optimizers. | In <i>arXiv preprint arXiv:2309.03409</i> . | | |
| 983 | | | | |
| 984 | 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 <i>Advances in Neural Information Processing Systems (NeurIPS)</i> . | | |
| 985 | | | | |
| 986 | Murat Yilmaz, Rory V O’Connor, and Paul Clarke. 2012. A Systematic Approach to the Comparison of Roles in the Software Development Processes. | In <i>International Conference on Software Process Improvement and Capability Determination</i> , pages 198–209. Springer. | | |
| 987 | | | | |
| 988 | Zhangyue Yin, Qiushi Sun, Cheng Chang, Qipeng Guo, Junqi Dai, Xuanjing Huang, and Xipeng Qiu. 2023. Exchange-of-Thought: Enhancing Large Language | | | |
| 989 | | | | |
| 990 | | | | |
| 991 | | | | |
| 992 | | | | |
| 993 | | | | |
| 994 | | | | |
| 995 | | | | |
| 996 | | | | |
| 997 | | | | |
| 998 | | | | |
| 999 | | | | |
| 1000 | | | | |
| 1001 | | | | |
| 1002 | | | | |
| 1003 | | | | |
| 1004 | | | | |
| 1005 | | | | |
| 1006 | | | | |
| 1007 | | | | |
| 1008 | | | | |
| 1009 | | | | |
| 1010 | | | | |
| 1011 | | | | |
| 1012 | | | | |
| 1013 | | | | |
| 1014 | | | | |
| 1015 | | | | |
| 1016 | | | | |
| 1017 | | | | |
| 1018 | | | | |
| 1019 | | | | |
| 1020 | | | | |
| 1021 | | | | |
| 1022 | | | | |
| 1023 | | | | |
| 1024 | | | | |
| 1025 | | | | |
| 1026 | | | | |
| 1027 | | | | |
| 1028 | | | | |
| 1029 | | | | |
| 1030 | | | | |
| 1031 | | | | |
| 1032 | | | | |
| 1033 | | | | |
| 1034 | | | | |
| 1035 | | | | |
| 1036 | | | | |
| 1037 | | | | |
| 1038 | | | | |
| 1039 | | | | |
| 1040 | | | | |
| 1041 | | | | |
| 1042 | | | | |
| 1043 | | | | |

- 1047 Model Capabilities through Cross-Model Communi-
1048 cation. In *Proceedings of the 2023 Conference on*
1049 *Empirical Methods in Natural Language Processing*
1050 (*EMNLP*), pages 15135–15153.
- 1051 An Zhang, Leheng Sheng, Yuxin Chen, Hao Li, Yang
1052 Deng, Xiang Wang, and Tat-Seng Chua. 2023a. **On**
1053 **Generative Agents in Recommendation**. In *arXiv*
1054 *preprint arXiv:2310.10108*.
- 1055 Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu,
1056 Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang,
1057 Yulong Chen, et al. 2023b. **Siren’s Song in the AI**
1058 **Ocean: A Survey on Hallucination in Large Lan-**
1059 **guage Models**. In *arXiv preprint arXiv:2309.01219*.
- 1060 Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu
1061 Lin, Yong-Jin Liu, and Gao Huang. 2023. **ExpeL:**
1062 **LLM Agents Are Experiential Learners**. In *arXiv*
1063 *preprint arXiv:2308.10144*.
- 1064 Tianming Zhao, Chunyang Chen, Yuanning Liu, and
1065 Xiaodong Zhu. 2021. **GUIGAN: Learning to Gener-**
1066 **ate GUI Designs Using Generative Adversarial Net-**
1067 **works**. In *IEEE/ACM International Conference on*
1068 *Software Engineering (ICSE)*, pages 748–760.
- 1069 Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou,
1070 Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan
1071 Bisk, Daniel Fried, Uri Alon, et al. 2023a. **We-**
1072 **bArena: A Realistic Web Environment for Build-**
1073 **ing Autonomous Agents**. In *arXiv preprint*
1074 *arXiv:2307.13854*.
- 1075 Wangchunshu Zhou, Yuchen Eleanor Jiang, Long Li,
1076 Jialong Wu, Tiannan Wang, Shi Qiu, Jintian Zhang,
1077 Jing Chen, Ruipu Wu, Shuai Wang, Shiding Zhu, Jiyu
1078 Chen, Wentao Zhang, Ningyu Zhang, Huajun Chen,
1079 Peng Cui, and Mrinmaya Sachan. 2023b. **Agents: An**
1080 **Open-source Framework for Autonomous Language**
1081 **Agents**. In *arXiv preprint arXiv:2309.07870*.

Appendix

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

A Notations

Table 5 presents a comprehensive list of all symbol notations employed in this paper, encompassing those utilized in the formalized description of the methodology as well as in the ablation experiments.

| Notation | Definition |
|-----------------------|--------------------------|
| \mathcal{C} | Chat Chain |
| \mathcal{P} | Phase |
| \mathcal{T} | Subtask |
| \mathcal{C} | Multi-turn Communication |
| τ | Solution Extraction |
| P | Prompt |
| ρ | Role Customization |
| \mathcal{I} | Instructor Agent |
| \mathcal{A} | Assistant Agent |
| \rightarrow | Instructing |
| \rightsquigarrow | Responding |
| \mathcal{M} | Working Memory |
| $\tilde{\mathcal{M}}$ | Episodic Memory |
| \leq | Until |
| \searrow | Removing |

Table 5: Main notations used in this paper.

B Case Study

This section presents a case study on software developed using ChatDev, detailing each stage of its lifecycle. We visualize ChatDev as a "digital software company" in Figure 5 and illustrate how ChatDev progresses through design, coding, testing, and ultimately completing the software. The representative software is "*Business Sales Performance Tracker*" with a user's requirement: "*Business Sales Performance Tracker* is a software application that helps businesses track and analyze their sales performance. It provides features for inputting sales data, generating reports, and visualizing sales performance metrics. The application also allows businesses to set sales goals and compare actual performance against targets".

Figure 6 illustrates the Business Sales Performance Tracker's user interface. On the top left, a data entry interface is displayed, where users can in-

put sales-related information. This interface allows for the repeated entry of customer names, product names, and sales figures into designated fields. Users can then click the "Add Sales Data" button to integrate this information into the tracking system. For generating comprehensive reports, the user can click the "Generate Report" button. This action produces a statistical report within a terminal window, displaying key metrics such as total revenue, sales growth, conversion rate, average order value, customer acquisition cost, and customer lifetime value. Additionally, a visual report in the form of a histogram is displayed on the right side of the window. The software includes tools in the toolbar, which enable users to customize the histogram's layout and style. These tools also provide options to save and export the graphical data representations.

Figures 7, 8, 9, 10, 11, 12, 13, 14, and 15 provide a comprehensive view of the multi-agent communication throughout various stages of the software development process. These stages include Design, Coding, Review, and Testing. Each figure captures the detailed dialogue and interactions that occur during these phases, showcasing the collaborative efforts and methodologies employed in the development of the software.

For screenshots of other examples of software that ChatDev has been able to build, see Figure 16.

C The SRDD Dataset

We have developed a comprehensive database containing 1,200 software descriptions for automatic software generation. The dataset structure is shown in Figure 17. The construction of this database adhered to the following three-stage strategy for constructing a diverse and unique dataset:

1. Random Sampling: First, ChatGPT is independently inquired multiple times to obtain software information under a certain category, and then the duplication is removed at the token granularity of the software name.
2. Sequential Sampling: Then we add the generated software information in sequence in the form of negative prompts, requiring ChatGPT to continue generating unique software information.
3. Check: Although ChatGPT has been required to follow certain rules when generating, LLM is more likely to be overconfident when generating according to rules than when judging

1159 based on rules. Therefore, our last step is to
1160 let ChatGPT determine whether the generated
1161 software follows the rules.

1162 This strategy initially establishes datasets by ran-
1163 dom sampling some software data, then records
1164 existing data, granting ChatGPT autonomy to pro-
1165 duce novel entries. SRDD is created with human-
1166 designed rules that make the created software easy
1167 for researchers to evaluate, for example, the col-
1168 lected software does not need internet or multi-
1169 player participation. The length distribution of soft-
1170 ware descriptions in SRDD is shown in Figure 18.

1171 We sought to analyze the effects and semantic
1172 features of the generated software descriptions by
1173 using t-SNE to perform dimensionality reduction
1174 and visualization on the description embedding
1175 generated by the OpenAI Ada Model.

1176 As demonstrated in figure 19, significant cluster-
1177 ing of tasks bearing the same color is observed. It
1178 can be concluded that 1) software descriptions of
1179 the same category are distributed in clusters, indi-
1180 cating that the generated descriptions are highly re-
1181 lated to their categories. 2) Descriptions in different
1182 subcategories under the same category are clustered
1183 together, such as the game subcategories in the
1184 lower right corner. 3) Some subcategories of differ-
1185 ent categories also show overlaps in the figure, such
1186 as Tools&Utilities and Graphics, Schedule and
1187 Business, Sports and Sports Game. Such an over-
1188 lap is comprehensible given the multi-functionality
1189 of some software applications that may not be con-
1190 fined to a single classification.

1191 This dataset will be made available under the CC-
1192 BY-NC-4.0 license, allowing all users free access
1193 and use for research purposes.

1194 D Software Statistic

1195 Figure 20 gives the statistical distribution of Chat-
1196 Dev generated software on the SRDD dataset. We
1197 give the distribution on software generation du-
1198 ration, the number of tokens that interacted with
1199 LLM, the number of files in the software, and the
1200 number of code lines.

1201 E Language Transition

1202 Natural language predominates during system de-
1203 sign, while programming language dominates post-
1204 design phases for development and testing. Explor-
1205 ing utterances in the design phase reveals four pri-
1206 mary communication modes: proposition (9.02%),

1207 agreement (50.66%), suggestion (25.99%), and de-
1208 cision (14.32%).

1209 The communications mainly center around four
1210 key topics: discussion organization, architecture
1211 design, social obligation, and dialogue summariza-
1212 tion. Agents also occasionally touch on topics like
1213 data security, technical terminology, and multiuser
1214 engagement, although these constitute a smaller
1215 part of the communication.

1216 To further analyze the natural language commu-
1217 nication topic of the agent in the design stage, we
1218 conducted a semi-supervised topic modeling based
1219 on Corex (Gallagher et al., 2017) on its dialogue
1220 data. The dialogue topics are divided into five parts:
1221 Start, Discussion Organization, Architecture De-
1222 sign, Social Obligation, and Summarization. The
1223 transition chart (Mezza et al., 2018) of the dialogue
1224 topics is shown in Figure 21. As shown in the fig-
1225 ure, basically the agent promotes the discussion in
1226 the design phase according to the above five steps.
1227 They focus on several rounds of discussion in archi-
1228 tecture design and social obligation (represented
1229 by the self-loop of the nodes with high probability
1230 in the figure). Sometimes the agent will skip some
1231 parts or return to the previous part to continue the
1232 discussion. This shows that the agent can organize
1233 informative dialogue during autonomous commu-
1234 nication, and naturally shift the discussion topic to
1235 the problem to be solved.

1236 F Requirement Implementation Analysis

1237 In this section, we explore the effectiveness of Chat-
1238 Dev in accurately transforming user requirements
1239 into functional software through a unique "demand
1240 back-translation" experiment. This experiment is
1241 designed to test the fidelity of ChatDev in realizing
1242 software that aligns with user requirements, us-
1243 ing a process akin to back-translation in language
1244 translation. The experiment begins with a docu-
1245 mentation phase, where the agent is tasked with
1246 writing a comprehensive software document solely
1247 based on the code, without any access to the origi-
1248 nal user requirements. This approach ensures that
1249 the agent's understanding and interpretation of the
1250 software's functionality is derived entirely from
1251 the code. After the documentation stage, we ap-
1252 ply an LLM-check and human-recheck process to
1253 determine the extent to which the user's require-
1254 ments have been successfully incorporated into the
1255 document. This evaluation is then followed by a
1256 thorough review conducted by human experts, who

cross-check the findings to ensure accuracy and completeness. This process mirrors the concept of back-translation in machine translation, where content is translated from a source language to a target language and then translated back to the source language. Similarly, in this context, ChatDev translates natural language requirements into programming code, which is then translated back into a natural language document. This method serves as a more straightforward approach compared to traditional consistency metrics, as it involves a comparison within the same linguistic framework — from natural language requirements to natural language documentation. The results, as presented in Table 6, demonstrate that through a robust chain of communication and multiple iterative phases, ChatDev has successfully adhered to and executed the user’s commands. The experiment reveals that ChatDev has proficiently developed software that accurately fulfills the user’s specified needs, thereby showcasing its effectiveness in requirement implementation.

G Supplementary Analysis

In this section, we delve into detailed comparative analyses, as represented in various tables, highlighting ChatDev’s performance relative to baseline models across a range of metrics. The key information or findings are as follows:

Tables 7 and 8 detail the results comparing the thoroughness of completeness between ChatDev and the baseline models. The outcomes assessing the executability of the solutions are shown in Tables 9 and 10, focusing on the executability aspect. Tables 11 and 12 present the results in terms of consistency, measuring how consistently ChatDev and its counterparts perform. The overall quality of the outputs, considering various factors, is encapsulated in Tables 13 and 14. In these tables, the color coding (red for ChatDev superiority and green for the opposite) visually demonstrates that ChatDev outperforms the baselines in almost all the evaluated categories, indicating its significant advantage.

In addition to computational experiments, to further validate the real-world practice of software evaluation, we compensated 30 human participants for their involvement in a real-name experiment assessment. The participant demographic comprises one associate professor, ten doctoral candidates, ten graduate students, and nine undergraduate students,

all hailing from prestigious universities, with a gender distribution of 5:2. Each participant was assigned the source code of 200 software pairs (rather than screenshots), one generated by our method and the other by a baseline method. Note, to avoid assessment errors, the 200 software pairs for different evaluators may overlap, and the final win rate is determined by taking the average. Moreover, to mitigate possible positional bias, and human participants independently assessed the task solutions, randomized to prevent order bias. Except for the ground-truth answers, the above description of the experimental setup was known to the participants. The experiment commenced only after obtaining consent from all participants. The main experimental results have been shown and analyzed in the main text. Furthermore, in Table 2, the results also indicate a variance in win rate distribution between human and GPT-4 evaluations. Human participants, capable of discerning nuanced differences in software solutions, are more critical, while GPT-4 shows a conservative approach, possibly due to its difficulty in assessing fine details in logic-intensive software comparisons.

H Framework Implementation

This section describes how we designed the three-layer architecture of the underlying implementation of ChatDev, and how we provided a SaaS platform on top of the ChatDev architecture.

H.1 Architecture

ChatDev employs a three-layer architecture to achieve a robust, modular, and customizable system for natural language to software (NL2Software). It consists of a Task Solving layer, a Cooperation layer, and an Agent layer. As shown in Figure 22, the generation process begins with a natural language query from the user as input, proceeding to invoke underlying layers, up to querying LLMs to obtain the raw response. The response is then progressively returned to a higher-level layer and processed. It is finally assembled into finished software and returned to the user. Additionally, ChatDev has an Environment module to maintain and manage persistent external information. The Utterance Parser and Solution Storer interact with ChatDev on all three layers, enabling flexible read and write on files such as Codes and Docs, thus facilitating information transmission and enhancing the ability of ChatDev.

1356 H.1.1 Task-Solving Layer

1357 At the topmost level, the Task Solving Layer of
1358 ChatDev loads global configurations and orches-
1359 trates the entire workflow. This layer adopts a
1360 chain-like structure named Chat Chain, which is
1361 composed of several simple and composed phases.
1362 Each phase corresponds to a stage in the traditional
1363 software engineering waterfall model, such as de-
1364 mand analysis, code generation, and code review.
1365 In detail, each phase executes the following three
1366 steps:

- 1367 • Retrieve the necessary contextual information
1368 from the environment based on the require-
1369 ments. For example, the code modification
1370 phase needs the context of the generated code,
1371 software requirements, and metadata.
- 1372 • Invoke the Cooperation layer to initiate the
1373 complete dialogue and collect conclusions
1374 based on the configuration and contextual in-
1375 formation.
- 1376 • Apply the conclusions as the output of the
1377 current phase and update this information in
1378 the environment module. For instance, the
1379 conclusion of the modification phase would
1380 be to review comments and modified codes.

1381 Each phase can be seen as an interactive process
1382 that reads and writes on the environment. Such
1383 modular design ensures that the entire chat chain
1384 can be flexibly orchestrated and phases can be
1385 reused according to requirements.

1386 H.1.2 Cooperation Layer

1387 Node execution within the chat chain is facilitated
1388 through LLM-based multi-agent communication.
1389 The Cooperation Layer initially configures the
1390 entire conversational context based on preset settings.
1391 It selects an instructor and an assistant from a role
1392 pool. The former role sets the task for the com-
1393 munication, while the latter role is responsible for
1394 achieving it.

1395 To accommodate communication needs of vary-
1396 ing complexity, the Cooperation Layer can config-
1397 ure loops and combinations for each phase's dia-
1398 logue, enabling what is called a Composed Phase.
1399 For instance, a Modification phase could be com-
1400 posed of a review comment communication and a
1401 review modification communication. In the review
1402 comment communication, the Programmer agent
1403 serves as the instructor and the Reviewer agent as

the assistant; in the review modification communica-
1404 tion, their roles would be swapped. Then these
1405 two communications are combined and repeated
1406 five times. The loops and combinations in the Com-
1407 posed Phase allow users to easily customize higher-
1408 order processes.

1409 Each phase is designed to have a clear objective
1410 which is to reach a conclusion and update the envi-
1411 ronment. However, controlling multi-round conver-
1412 sations between agents is challenging and does not
1413 always reach a consensus and generate conclusions.
1414 Therefore, ChatDev incorporates communication
1415 protocols within the Cooperation Layer to track the
1416 entire dialogue. With the protocol, ChatDev would
1417 terminate the communication when a conclusion
1418 pattern emerges. When the dialogue reaches the
1419 maximum round limit without a conclusion, Chat-
1420 Dev will activate a self-reflection function, which
1421 reads all the dialogue history from this communi-
1422 cation and designates a Counselor Agent to sum-
1423 marize a conclusion.

1425 H.1.3 Agent Layer

1426 Agent Layer processes each inquiry from the Com-
1427 munication Layer and constructs the final prompts
1428 that are sent to the Large Language Models. It real-
1429 izes standardized Agentization modules including
1430 Memorizing, Planning, and Tool Using for enhanc-
1431 ing the capacity of agents.

1432 Fluent task-oriented communication between
1433 two agents necessitates at least three points, which
1434 are "Who I am", "Who you are" and "What is
1435 our communication goal". All the information is
1436 conveyed to agents through prompt assembling.
1437 The prompts for agents include system background
1438 prompts, role-specific prompts for the conversa-
1439 tional participants, and phase-target prompts em-
1440 bedded with dynamic contextual information. The
1441 role prompts and other prompt data are exposed
1442 through configurable files, allowing users to cus-
1443 tomize their roles and apply them to agents without
1444 writing the code.

1445 Beyond automated communication capabilities,
1446 agents require various Agentization functions. For
1447 example, 1) each agent needs a memorizing module
1448 to retain a history of past dialogues and conclusions.
1449 2) The Designer Agent relies on the Planning mod-
1450 ule to transform user prompts into detailed require-
1451 ments and schedules. 3) The Reviewer and Tester
1452 Agents need an array of external programming
1453 tools such as Shell, Compiler, Git, and Docker for
1454 environmental interaction to manage, compile, and

improve code.

Solution Extractor This encourages the assistant to respond in a specified format with high probability, avoiding any deviation from the structure of the conversation, and preventing vague or incomplete responses. ChatDev, according to the corresponding protocol format, will terminate the communication and extract the corresponding information as the solution of the current phase.

When the two are talking endlessly (e.g., keep saying "thank you") and the termination protocol is not triggered, we force terminate the dialogue and adopt the self-reflection mechanism by feeding the whole phase dialogue to a Counselor agent and let it summarize the final information of the whole dialogue.

$$\psi(x) = \begin{cases} \text{RegParse}(x), & \text{If Protocol Triggered} \\ \text{SelfReflectionR}(x), & \text{Otherwise} \end{cases} \quad (8)$$

where x is the dialogue between agents, and *RegParse* denotes the Regular Expression Parse for extracting the conclusion from the response.

H.2 A SaaS Platform

In addition to open-source code, the ChatDev system also offers an easy-to-use SaaS platform that allows users to customize their "virtual software company" based on multi-agent communication without needing to understand the architecture details. The SaaS platform is designed with the following four guiding principles to meet the needs of users of all levels:

- **User-Friendly:** The SaaS Platform provides a comprehensive set of user-friendly web interfaces. Users can simply log into the website to use it instead of launching ChatDev locally. The web interface displays the entire conversation in a visualized, streaming manner. Besides, Users can track the progress of software generation through the chat chain progress bar.

- **Customizable:** Users can arrange the entire chat chain by dragging and dropping phase cards on the web GUI. Moreover, they can customize the roles of agents, the background prompts of chat chain nodes, and so on without coding. Besides generating code, ChatDev also supports the creation of images and other assets to accommodate highly personalized requirements.

- **Intervenable:** Fully automated processes might

inevitably introduce a small amount of bias, diverging from the user requirement. Therefore, ChatDev supports user intervention in two ways: 1) HAI (Human-Agent Interaction) mode: during the software testing phase, the user can take the control of tester agent and interact directly with the programmer agent, providing information about bugs or dissatisfaction and instructing the Programmer agent to make changes. 2) Incremental Development: users can request new features, allowing ChatDev to continue development on the completed software.

- **Safety:** On the one hand, ChatDev endows agents with the capability to use Git tools, enabling version management and backtracking. On the other hand, to prevent agents from exploiting tools to create security issues, ChatDev provides a Docker sandbox environment in which all code and script executions are carried out safely, avoiding potential harm to the user's system.

I Broader Impact

With the modular design of the chat chain, ChatDev can perform multi-agent communication with flexible configuration. It enables ChatDev to complete various complex applications in the field of software engineering and even more general domains. For phases such as code writing, code review, and system testing, ChatDev can perform advanced modifications and orchestration, including conditional branching and loop execution. These customizations allow ChatDev to adapt to the needs of a variety of software engineering tasks, like version iteration, incremental development, Git management, etc. For modifications to natural language communication in phases like Design, ChatDev can also facilitate applications such as brainstorming and requirements schedule. ChatDev can even execute a comprehensive data cleansing process with the integration of external tools like web crawlers and data analysis.

In this section, we exemplify the powerful extensibility of ChatDev by using version iteration as an illustrative case. A simple and clear way to extend the Chat Chain for version iteration is as follows,

1. In the design phase, agents are required to discuss three features with progressively increasing complexity, which are then extracted for features of the three new versions.
2. At the end of the chain, we add three groups

1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520

1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540

1541
1542
1543
1544
1545
1546
1547
1548
1549

of code review and system testing. The "comment" for each review is set to modify the code to satisfy the corresponding new version feature".

3. After system testing, the code is stored independently as a new version of the software.

Each software release builds on the previous version, with increasingly complex, advanced features.

Figure 23 24, 25 and 26 provides the statistical results for the baseline version and three new versions. It is observable that: 1) As version iteration progresses, functionality gradually becomes more complex, and the number of files and lines of code increase steadily, with consistent trends across different software categories. 2) The implementation of more complex features is accompanied by larger-scale modifications, and the frequency of code updates also continues to rise with each version iteration. 3) Although there is a slight decrease in the executability of software across different categories in higher versions, a relatively high degree of executability is still maintained.

J Prompts

Demands Classification In the design phase, we consolidated the discussed demands into 13 distinct perspectives, each accompanied by comprehensive definitions and illustrative examples. Subsequently, we prompted and human-rechecked to categorize the agent's analysis of demands for each software project into one or multiple of these perspectives. The elaborate classification prompt is depicted in Figure 27. The distribution of these classification results is represented in Figure 2.

Reviews Classification Employing a methodology akin to the demands classification, we prompted and human-rechecked by categorizing review comments into predefined review categories. The detailed prompt used for this classification is illustrated in Figure 28. This classification encompasses reviews from all rounds of feedback. Statistical analysis of these classifications is displayed in Figure 3.

Requirement Implementation We prompted and human-rechecked to verify whether the demands identified during the Design phase had been realized, based on the manual. Figure 29 shows the prompt employed for this purpose.

K Software and Data

The source code of the system and the datasets utilized in this paper are provided in `Software.zip` and `Data.zip`, respectively. These archives provide comprehensive configuration guidelines, command instructions for execution, examples of logs, and additional resources. These components are essential for assessing the reproducibility of our technology. Moreover, all these materials will be made publicly available to support subsequent academic research.

L AI Assistants

ChatGPT⁴ was used purely with the language of the paper during the writing process, including spell-checking and paraphrasing the authors' original content, without suggesting new content. Any content generated with the assistant underwent meticulous manual review and subsequently received final approval from the authors.

⁴<https://chat.openai.com/>



Figure 5: ChatDev, like a virtual software company, brings together LLM-powered "software agents" from diverse social identities, including professional programmers and test engineers. When presented with a preliminary task requirement by a human "client" (e.g., "develop a gomoku game"), the agents at ChatDev engage in effective communication and mutual verification through collaborative chatting, which enables them to automatically craft comprehensive software solutions.

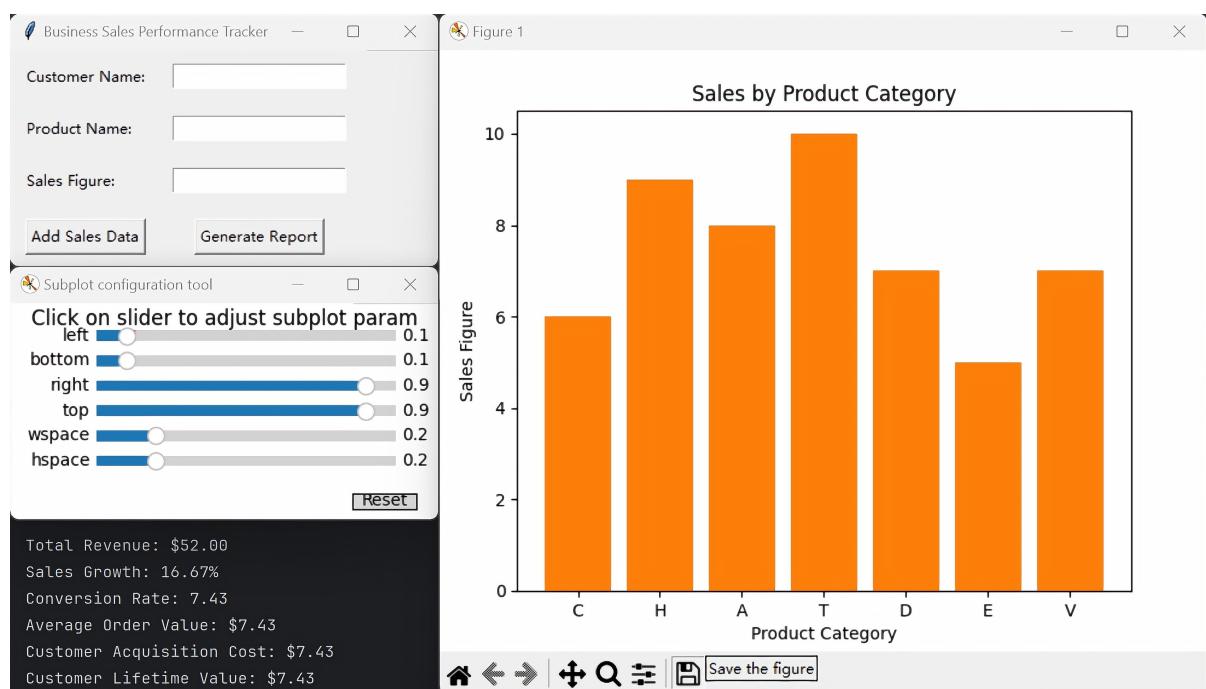


Figure 6: Demonstration of the "Business Sales Performance Tracker" software developed by ChatDev.

Design



: CEO (Chief Executive Officer)



: CTO (Chief Technology Officer)

Demand Analysis:



: To start the discussion, I propose that we prioritize simplicity and ease of use in the user interface design. We should aim to minimize the learning curve for users and ensure that the software is accessible to both tech-savvy and non-technical users. What are your thoughts on this approach? Please decide on whether we should...



: I agree with your proposal to... Prioritizing simplicity and ease of use in the user interface design will help us create a user-friendly experience that caters to both tech-savvy and non-technical users. It will contribute to increased efficiency, accuracy, and overall user satisfaction...

Brainstorming:



: Discussion Point: 1) Layout and Navigation: To begin, we need to decide on the layout and navigation of the software. Should we opt for a traditional menu-based navigation system or consider a more modern approach, such as a dashboard-style interface? What are your thoughts on this? ; 2) Visual Design: Next, let's discuss the visual design of the software. What color scheme and typography should we use to...

Self-Plan:



: Given the software description, we have decided to complete the task and have discussed several demands and ideas, such as:

Demands: 1) the software should prioritize simplicity and ease of use in the user interface design to ensure a user-friendly experience for businesses to input sales data effortlessly this includes... 2) the software should provide comprehensive reports that summarize sales performance metrics such as... 3) the software should allow businesses to set both short-term and long-term sales goals and provide visual indicators or notifications to...

Ideas: 1) idea: integration with CRM systems and e-commerce platforms integrating the "business sales performance tracker" software with... 2) idea: customizable reports and dashboards offering extensive customization options for ... 3) idea: real-time analytics and notifications ...

Your task is to extract the essential components from the given requirements and rewrite the task prompt to create an initial version of the software that is both easy to implement and meets the basic requirements. ...



: Design a software application called "Business Sales Performance Tracker" that enables businesses to effectively track and analyze their sales performance. The application should provide a user-friendly interface for inputting sales data, including customer information, product details, and sales figures. It should allow businesses to generate comprehensive reports that summarize sales performance metrics such as total revenue, sales growth, conversion rates, average order value, customer acquisition cost, customer lifetime value, and sales by product category. The software should also offer visualizations, such as charts, graphs, and tables, to help businesses visualize their sales performance trends over time...

Figure 7: Agent communication in the design phase.

Code Writing



: CTO (Chief Technology Officer)



: Programmer



: We have decided to complete the task through a executable software with multiple files implemented via Python. As the Programmer, to satisfy the new user's demands, you should write one or multiple files and make sure that every detail of the architecture is, in the end, implemented as code. The software should be equipped with graphical user interface (GUI) so that user can visually and graphically use it; so you must choose a GUI framework (e.g., in Python, you can implement GUI via tkinter, Pygame, Flexx, PyGUI, etc.).

Think step by step and reason yourself to the right decisions to make sure we get it right.

You will first lay out the names of the core classes, functions, methods that will be necessary, as well as a quick comment on their purpose.



main.py

```
import tkinter as tk
from tkinter import messagebox
from sales_data import SalesData
from sales_report import SaleReport
class SalesPerformanceTracker:
    def __init__(self, root):
        self.root = root
        self.root.title("BusinessSalesPerformanceTracker")
        self.sales_data = SalesData()
        self.create_widgets()
    def create_widgets(self):
        # Create and configure input fields
        self.customer_name_label = tk.Label(self.root, text="CustomerName:")
        self.customer_name_label.grid(row=0, column=0, padx=10, pady=10, sticky=tk.W)
        self.customer_name_entry = tk.Entry(self.root)
        self.customer_name_entry.grid(row=0, column=1, padx=10, pady=10)
        self.product_name_label = tk.Label(self.root, text="ProductName:")
        self.product_name_label.grid(row=1, column=0, padx=10, pady=10, sticky=tk.W)
        self.product_name_entry = tk.Entry(self.root)
        self.product_name_entry.grid(row=1, column=1, padx=10, pady=10)
        self.sales_figure_label = tk.Label(self.root, text="SalesFigure:")
        self.sales_figure_label.grid(row=2, column=0, padx=10, pady=10, sticky=tk.W)
        self.sales_figure_entry = tk.Entry(self.root)
        self.sales_figure_entry.grid(row=2, column=1, padx=10, pady=10)
    def add_sales_data(self):
        customer_name = self.customer_name_entry.get()
        product_name = self.product_name_entry.get()
        sales_figure = self.sales_figure_entry.get()
        if customer_name and product_name and sales_figure:    self.sales_data.
            add_data(customer_name, product_name, float(sales_figure))
            messagebox.showinfo("Success", "Salesdataaddedsuccessfully!")
        else:
            ...
...
```

Figure 8: Agent communication in the code writing (*i.e.*, coding) phase, part 1.

Code Writing(continued)



: Programmer

main.py(continued)

```
...
    else:
        messagebox.showerror("Error", "Please fill in all fields.")
    self.clear_input_fields()
def generate_report(self):
    report = SalesReport(self.sales_data)
    report.generate()
def clear_input_fields(self):
    self.customer_name_entry.delete(0, tk.END)
    self.product_name_entry.delete(0, tk.END)
    self.sales_figure_entry.delete(0, tk.END)
if __name__ == "__main__":
    root = tk.Tk()
    app = SalesPerformanceTracker(root)
    root.mainloop()
```

sales_data.py

```
class SalesData:
    def __init__(self):
        self.data = []
    def add_data(self, customer_name, product_name, sales_figure):
        self.data.append({
            "customer_name": customer_name,
            "product_name": product_name,
            "sales_figure": sales_figure
        })
    def get_total_revenue(self):
        return sum(data["sales_figure"] for data in self.data)
    def get_sales_growth(self):
        if len(self.data) < 2:
            return 0
        first_sales = self.data[0]["sales_figure"]
        last_sales = self.data[-1]["sales_figure"]
        return (last_sales - first_sales) / first_sales * 100
    def get_conversion_rate(self):
        if len(self.data) == 0:
            return 0
        total_sales = sum(data["sales_figure"] for data in self.data)
        total_customers = len(self.data)
        return total_sales / total_customers
    def get_average_order_value(self):
        if len(self.data) == 0:
            return 0
        total_sales = sum(data["sales_figure"] for data in self.data)
        total_orders = len(self.data)
        return total_sales / total_orders
    def get_customer_acquisition_cost(self):
        if len(self.data) == 0:
            return 0
        total_sales = sum(data["sales_figure"] for data in self.data)
        total_customers = len(self.data)
        return total_sales / total_customers
    def get_customer_lifetime_value(self):
...
...
```

Figure 9: Agent communication in the code writing phase, part 2.

Code Writing(continued)



: Programmer

sales_data.py(continued)

```
...
def get_customer_lifetime_value(self):
    if len(self.data) == 0:
        return 0
    total_sales = sum(data["sales_figure"] for data in self.data)
    total_customers = len(self.data)
    return total_sales / total_customers
def get_sales_by_product_category(self):
    sales_by_category = {}
    for data in self.data:
        product_name = data["product_name"]
        sales_figure = data["sales_figure"]
        if product_name in sales_by_category:
            sales_by_category[product_name] += sales_figure
        else:
            sales_by_category[product_name] = sales_figure
    return sales_by_category
```

sales_report.py.py

```
import matplotlib.pyplot as plt
class SalesReport:
    def __init__(self, sales_data):
        self.sales_data = sales_data
    def generate(self):
        pass
    def generate_summary_report(self):
        total_revenue = self.sales_data.get_total_revenue()
        sales_growth = self.sales_data.get_sales_growth()
        conversion_rate = self.sales_data.get_conversion_rate()
        average_order_value = self.sales_data.get_average_order_value()
        customer_acquisition_cost = self.sales_data.get_customer_acquisition_cost()
        customer_lifetime_value = self.sales_data.get_customer_lifetime_value()
        report = f"SummaryReport:\n\n"
        report += f"TotalRevenue: ${total_revenue:.2f}\n"
        report += f"SalesGrowth: {sales_growth:.2f}%\n"
        report += f"ConversionRate: {conversion_rate:.2f}\n"
        report += f"AverageOrderValue: ${average_order_value:.2f}\n"
        report += f"CustomerAcquisitionCost: ${customer_acquisition_cost:.2f}\n"
        report += f"CustomerLifetimeValue: ${customer_lifetime_value:.2f}\n"
        print(report)
    def generate_sales_by_product_category_report(self):
        sales_by_category = self.sales_data.get_sales_by_product_category()
        categories = list(sales_by_category.keys())
        sales_figures = list(sales_by_category.values())
        plt.bar(categories, sales_figures)
        plt.xlabel("ProductCategory")
        plt.ylabel("SalesFigure")
        plt.title("SalesbyProductCategory")
        plt.show()
```



: The main.py file contains the main application class `SalesPerformanceTracker`, which handles the GUI and user interactions. It imports the `SalesData` class from sales_data.py and the `SalesReport` class from sales_report.py. The `add_sales_data` method is responsible for...

Figure 10: Agent communication in the code writing phase, part 3.

Code Complete



: CTO (Chief Technology Officer)



: Programmer



Unimplemented File

`sales_report.py`

In our software, each file must strictly follow a markdown code block format, where the following tokens must be replaced such that "FILENAME" is the lowercase file name including the file extension, "LANGUAGE" in the programming language, "DOCSTRING" is a string literal specified in source code that is used to document a specific segment of code, and "CODE" is the original code:

FILENAME
`` LANGUAGE
''
DOCSTRING
''
CODE
''

As the Programmer, to satisfy the complete function of our developed software, you have to implement all methods in the `sales_report.py` file which contains an unimplemented class. Now, implement all methods of the `sales_report.py` and all other codes needed, then output the fully implemented codes, strictly following the required format.



`sales_report.py`

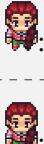
```
...  
def __init__(self, sales_data):  
    self.sales_data = sales_data  
def generate(self):  
    self.generate_summary_report()  
    self.generate_sales_by_product_category_report()  
def generate_summary_report(self):  
    ...
```

Code Diff on `sales_report.py`

```
def generate(self):  
-   pass  
+   self.generate_summary_report()  
+   self.generate_sales_by_product_category_report()
```

Figure 11: Agent communication in the code complete phase.

Code Review (Round 1)



: Reviewer



: Programmer

{Codes of current version}

<Comment>

The highest priority comment is that the `get_customer_lifetime_value` method in the `SalesData` class is currently returning the same value as the `get_customer_acquisition_cost` method. This is incorrect as the customer lifetime value should be calculated differently.

To fix this, you need to update the logic in the `get_customer_lifetime_value` method to calculate the average sales per customer over the lifetime of the business. You can do this by dividing the total revenue by the total number of customers.

Here's the revised code for the `get_customer_lifetime_value` method:

`sales_data.py`

```
def get_customer_lifetime_value(self):
    if len(self.data) == 0:
        return 0
    total_sales = sum(data["sales_figure"] for data in self.data)
    total_customers = len(set(data["customer_name"] for data in self.data))
    return total_sales / total_customers
```

Please make this change and ensure that the `get_customer_lifetime_value` method returns the correct value.



`sales_data.py`

```
...
def get_customer_acquisition_cost(self):
    if len(self.data) == 0:
        return 0
    total_sales = sum(data["sales_figure"] for data in self.data)
    total_customers = len(self.data)
    return total_sales / total_customers
def get_customer_lifetime_value(self):
    if len(self.data) == 0:
        return 0
    total_sales = sum(data["sales_figure"] for data in self.data)
    total_customers = len(set(data["customer_name"] for data in self.data))
    return total_sales / total_customers
...
```

Code Diff on `sales_data.py`

```
def get_customer_lifetime_value(self):
    if len(self.data) == 0:
        return 0
    total_sales = sum(data["sales_figure"] for data in self.data)
-   total_customers = len(self.data)
+   total_customers = len(set(data["customer_name"] for data in self.data))
```

Figure 12: Agent communication in the code review phase, round 1. To save space, we have omitted the specific code with {Codes of current version}.

Code Review (Round 2)



: Reviewer



: Programmer



{Codes of current version}

The highest priority comment is that the GUI elements of buttons for "Add Sales Data" and "Generate Report" are not implemented in the **SalesPerformanceTracker** class in main.py. To make the software user-friendly and provide intuitive controls for shape manipulation, the GUI elements need to be created.

To fix this, you need to implement the GUI layout and controls in the **SalesPerformanceTracker** class.

For example:

main.py

```
self.add_button = tk.Button()  
self.add_button.grid()  
self.generate_report_button = tk.Button()  
self.generate_report_button.grid()
```

Make sure to replace the placeholder code with the actual implementation of the GUI elements and their corresponding event bindings and callback functions. Once these changes are made, the software will have a user-friendly interface with intuitive controls.



main.py

```
...  
  
self.sales_figure_entry = tk.Entry(self.root)  
self.sales_figure_entry.grid(row=2, column=1, padx=10, pady=10)  
# Create buttons  
self.add_button = tk.Button(self.root, text="AddSalesData", command=self.  
    add_sales_data)  
self.add_button.grid(row=3, column=0, padx=10, pady=10)  
self.generate_report_button = tk.Button(self.root, text="GenerateReport", command=  
    self.generate_report)  
self.generate_report_button.grid(row=3, column=1, padx=10, pady=10)  
def add_sales_data(self):  
    customer_name = self.customer_name_entry.get()  
    product_name = self.product_name_entry.get()  
  
...
```

Code Diff on main.py

```
self.sales_figure_entry = tk.Entry(self.root)  
self.sales_figure_entry.grid(row=2, column=1, padx=10, pady=10)  
+ # Create buttons  
+ self.add_button = tk.Button(self.root, text="AddSalesData", command=self.  
    add_sales_data)  
+ self.add_button.grid(row=3, column=0, padx=10, pady=10)  
+ self.generate_report_button = tk.Button(self.root, text="GenerateReport", command=  
    self.generate_report)  
+ self.generate_report_button.grid(row=3, column=1, padx=10, pady=10)
```

Figure 13: Agent communication in the code review phase, round 2. To save space, we have omitted the specific code with {Codes of current version}.

System Testing (Round 1)



: Tester



: Programmer



Test Reports of Source Codes:

Test Error on sales_report.py

```
Traceback (most recent call last):
  File "sales_report.py", line 6, in <module>
    import matplotlib.pyplot as plt
ModuleNotFoundError: No module named 'matplotlib'
```

Error Summary of Test Reports: "nothing need to do on codes, install the missing module"

Note that each file must strictly follow Now, use the format exemplified above and modify the problematic codes based on the error summary. Output the codes that you fixed based on the test reported and corresponding explanations (strictly follow the format defined above, including FILENAME, LANGUAGE, DOCSTRING and CODE; incomplete "TODO" codes are strictly prohibited). If no bugs are reported, please return only one line like "<INFO> Finished".



<Action> Install matplotlib package

```
> pip3 install matplotlib
Collecting matplotlib
...
Downloaded matplotlib-3.8.2-cp39-cp39-macosx_11_0_arm64.whl (7.5 MB)
    7.5/7.5 MB 7.6 MB/s eta 0:00:00
Installing collected packages: matplotlib
Successfully installed matplotlib-3.8.2
```

sales_report.py

```
import matplotlib.pyplot as plt
class SalesReport:
    def __init__(self, sales_data):
        self.sales_data = sales_data
    ...
    
```

Figure 14: Agent communication in the system testing phase, round 1.

System Testing (Round 2)



: Tester



: Programmer



Test Error on main.py

```
Test Reports of Source Codes:  
Traceback (most recent call last):  
  File "main.py", line 9, in <module>  
    from sales_report import SaleReport  
ImportError: cannot import name 'SaleReport' from 'sales_report' (sales_report.py)
```

Error Summary of Test Reports:

Based on the test report, the bug causing the problem is an import error in the main.py file. The error message states that it cannot import the name `SaleReport` from the `sales_report` module. This suggests that there is an issue with the import statement in the main.py file.

To fix this bug, you need to import the correct name from the `sales_report` module.

Note that each file must strictly follow Now, use the format exemplified above and modify the problematic codes based on the error summary. Output the codes that you fixed based on the test reported and corresponding explanations (strictly follow the format defined above, including FILENAME, LANGUAGE, DOCSTRING and CODE; incomplete "TODO" codes are strictly prohibited). If no bugs are reported, please return only one line like "<INFO> Finished".



main.py

```
...  
Business Sales Performance Tracker  
Author: ChatDev  
...  
import tkinter as tk  
from tkinter import messagebox  
from sales_data import SalesData  
from sales_report import SalesReport  
class SalesPerformanceTracker:  
    def __init__(self, root):  
        self.root = root  
...  
...
```

Code Diff on main.py

```
  from sales_data import SalesData  
- from sales_report import SaleReport  
+ from sales_report import SalesReport
```

Figure 15: Agent communication in the system testing phase, round 2.

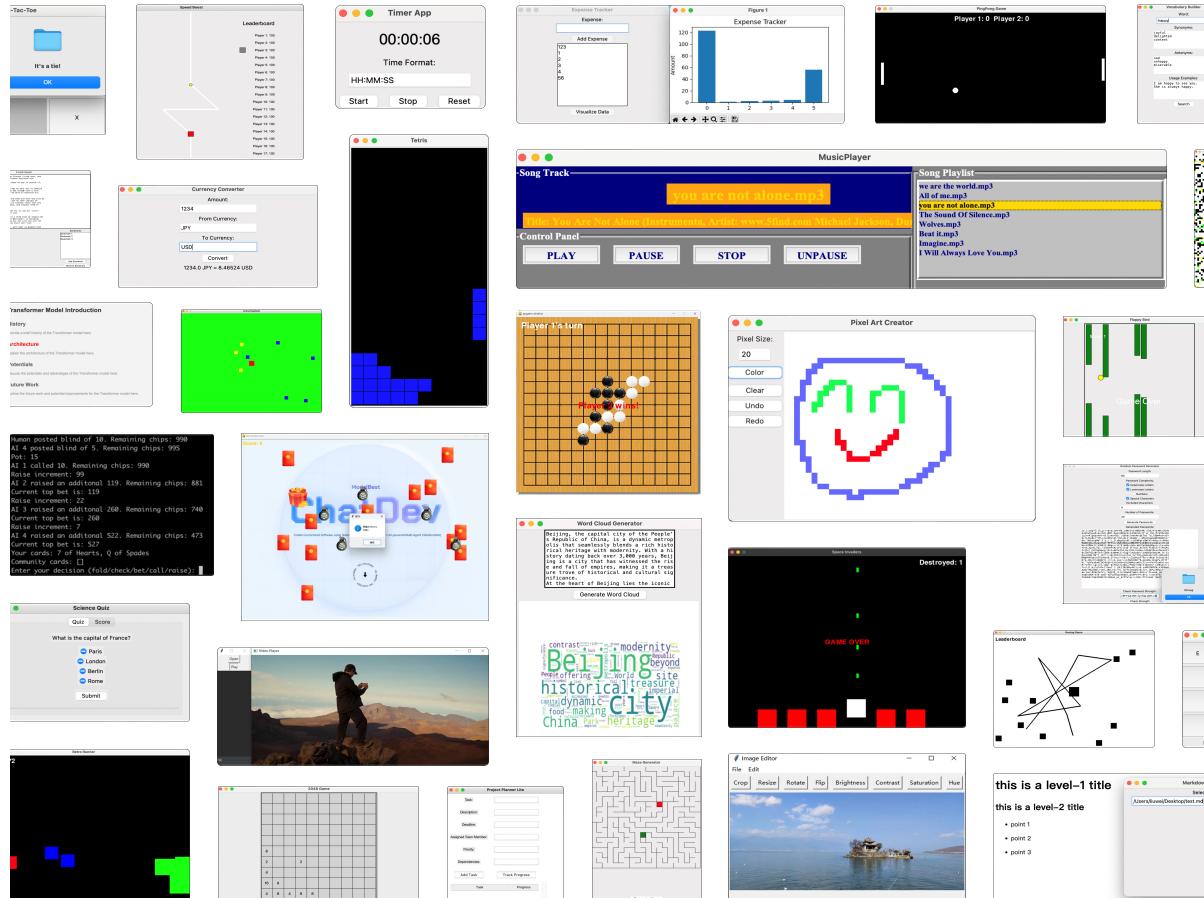


Figure 16: The software repository crafted by ChatDev encompasses a diverse array of software categories, including but not limited to the game category and tool category. Each category contains a range of applications, each uniquely designed to meet specific user requirements and functionalities. The game category includes a variety of games developed using ChatDev, ranging from simple puzzle games to more complex strategy and simulation games. These games are designed not only for entertainment but also to demonstrate the capabilities of ChatDev in handling intricate logic, graphics, and user interaction. The tool category comprises various utility and productivity tools. Examples might include applications for data analysis, task management, or content creation. These tools are tailored to enhance productivity and efficiency, showcasing ChatDev's ability to create software that addresses practical, everyday needs. In addition to these categories, the ChatDev-created software warehouse likely includes many other types of software, each illustrating the versatility and breadth of applications that can be developed using this advanced development platform.

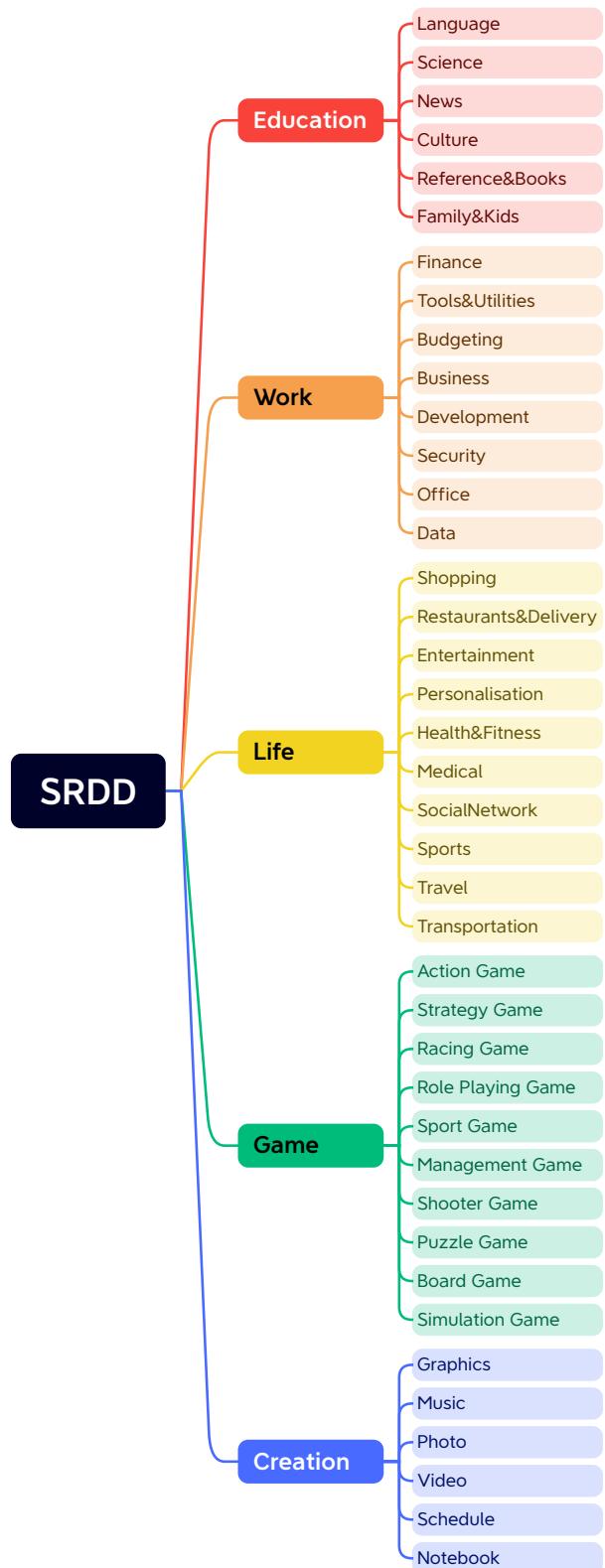


Figure 17: The hierarchy of the SRDD dataset.

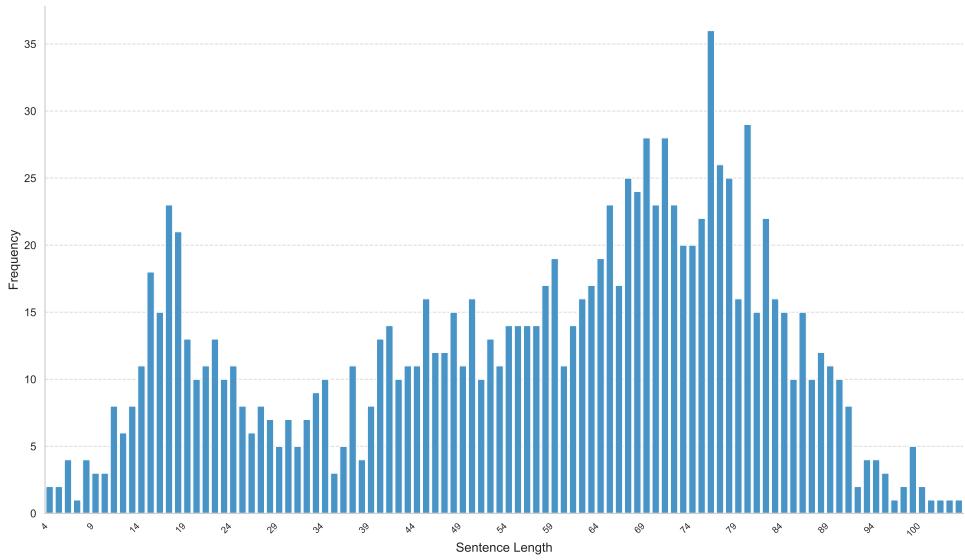


Figure 18: The software description length distribution in SRDD.

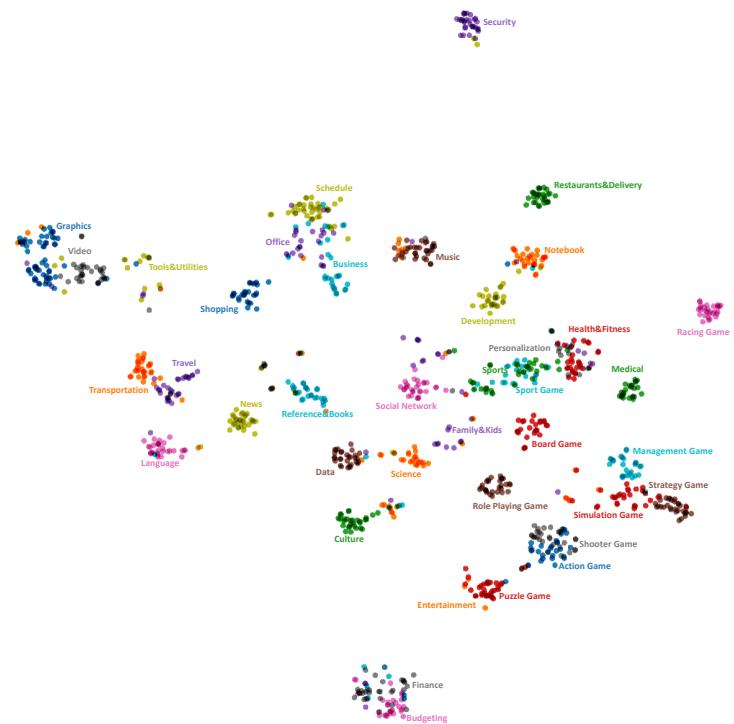


Figure 19: The t-SNE visualization of the SRDD task embeddings via the ada embedding model.

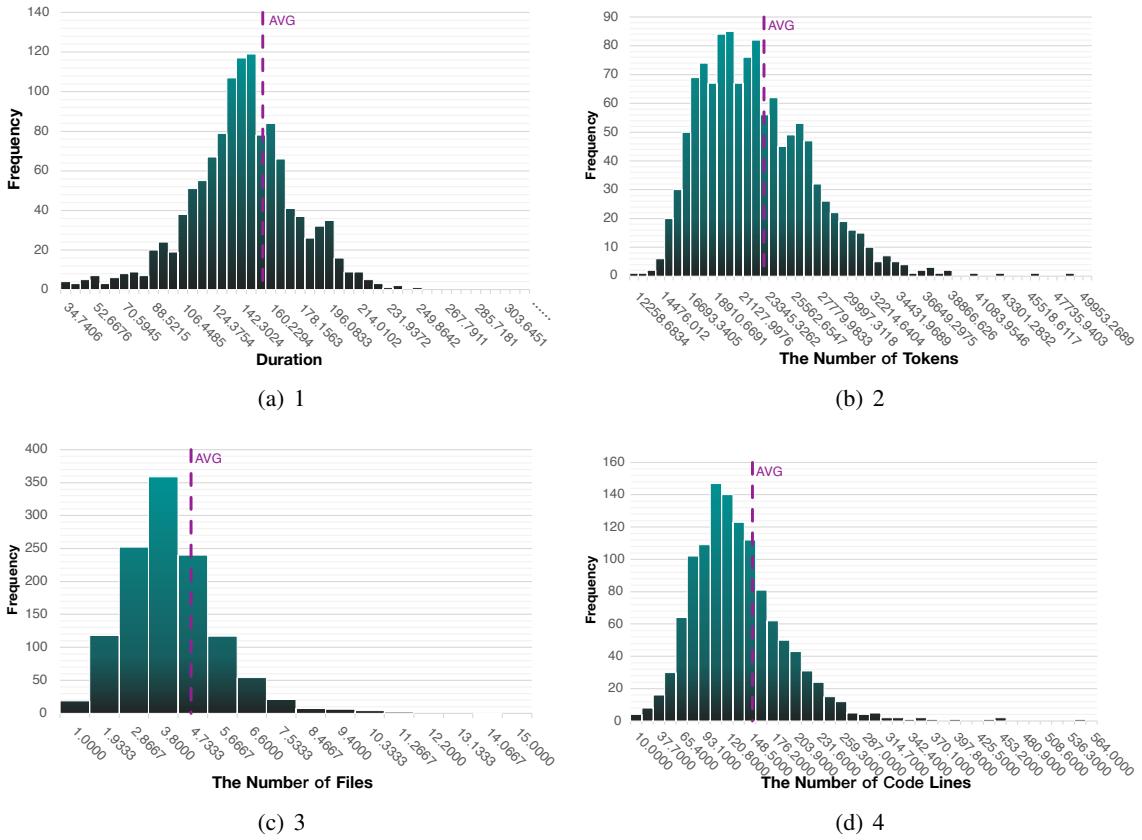


Figure 20: Statistical data distribution, including the duration taken, the number of tokens utilized, the number of generated code files, and total lines of code across all files.

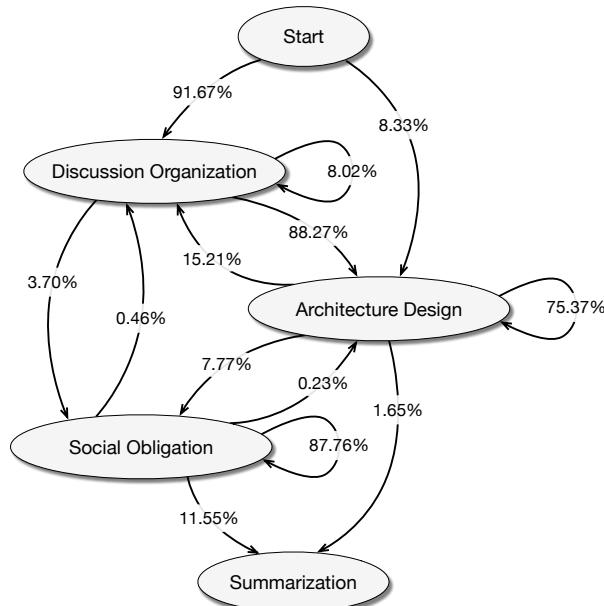


Figure 21: The transition graph in the design phase.

| Primary Category | Secondary Category | Requirement Implementation Ratio | | AVG #Demand Points |
|------------------|---------------------|----------------------------------|--------|-----------------------|
| | | AVG | VAR | |
| Education | Language | 0.8028 | 0.0143 | 6.79 |
| | Science | 0.8333 | 0.0177 | 7.30 |
| | News | 0.7713 | 0.0235 | 6.73 |
| | Culture | 0.7233 | 0.0280 | 6.20 |
| | ReferenceBooks | 0.7823 | 0.0229 | 6.20 |
| | FamilyKids | 0.7493 | 0.0383 | 7.70 |
| Work | Finance | 0.8330 | 0.0057 | 6.43 |
| | ToolsUtilities | 0.8580 | 0.0078 | 7.73 |
| | Budgeting | 0.7490 | 0.0262 | 6.97 |
| | Business | 0.7993 | 0.0115 | 6.40 |
| | Development | 0.8143 | 0.0113 | 6.47 |
| | Security | 0.7910 | 0.0225 | 6.57 |
| | Office | 0.7627 | 0.0270 | 6.93 |
| | Data | 0.7797 | 0.0127 | 7.20 |
| Life | Shopping | 0.7847 | 0.0331 | 7.00 |
| | RestaurantsDelivery | 0.8047 | 0.0193 | 7.20 |
| | Entertainment | 0.8067 | 0.0170 | 7.00 |
| | Personalisation | 0.8287 | 0.0087 | 6.53 |
| | HealthFitness | 0.7597 | 0.0259 | 6.93 |
| | Medical | 0.7440 | 0.0326 | 5.70 |
| | SocialNetwork | 0.7367 | 0.0327 | 6.50 |
| | Sports | 0.7957 | 0.0220 | 6.73 |
| | Travel | 0.8063 | 0.0145 | 6.47 |
| | Transportation | 0.7857 | 0.0312 | 6.93 |
| Game | ActionGame | 0.7443 | 0.0379 | 7.93 |
| | StrategyGame | 0.7363 | 0.0371 | 6.83 |
| | RacingGame | 0.7433 | 0.0299 | 6.67 |
| | RolePlayingGame | 0.8043 | 0.0103 | 6.50 |
| | SportGame | 0.7497 | 0.0239 | 6.43 |
| | ManagementGame | 0.7583 | 0.0401 | 7.50 |
| | ShooterGame | 0.7117 | 0.0281 | 7.97 |
| | PuzzleGame | 0.8260 | 0.0187 | 7.77 |
| | BoardGame | 0.7893 | 0.0286 | 5.97 |
| | SimulationGame | 0.7327 | 0.0275 | 6.50 |
| Creation | Graphics | 0.8340 | 0.0181 | 7.70 |
| | Music | 0.8293 | 0.0066 | 7.00 |
| | Photo | 0.8140 | 0.0169 | 7.13 |
| | Video | 0.8427 | 0.0119 | 7.40 |
| | Schedule | 0.7853 | 0.0270 | 7.83 |
| | Notebook | 0.7203 | 0.0407 | 7.10 |
| All | All | 0.7831 | 0.0227 | 6.92 |

Table 6: Requirement Implementation Ratio.

| Primary Category | Secondary Category | Completeness | | |
|------------------|---------------------|--------------|---------|----------|
| | | GPT-Engineer | ChatDev | Δ |
| Life | Entertainment | 0.3929 | 0.6319 | 0.2390 |
| | HealthFitness | 0.4667 | 0.7267 | 0.2600 |
| | Medical | 0.7000 | 0.6951 | -0.0049 |
| | Personalisation | 0.5000 | 0.3791 | -0.1209 |
| | RestaurantsDelivery | 0.4333 | 0.4739 | 0.0406 |
| | Shopping | 0.4000 | 0.4739 | 0.0739 |
| | SocialNetwork | 0.3000 | 0.5687 | 0.2687 |
| | Sports | 0.6000 | 0.6003 | 0.0003 |
| | Transportation | 0.4333 | 0.4424 | 0.0091 |
| | Travel | 0.4000 | 0.4739 | 0.0739 |
| Work | Budgeting | 0.6667 | 0.7899 | 0.1232 |
| | Business | 0.5333 | 0.5687 | 0.0354 |
| | Data | 0.4000 | 0.6635 | 0.2635 |
| | Development | 0.5667 | 0.6951 | 0.1284 |
| | Finance | 0.5333 | 0.7267 | 0.1934 |
| | Office | 0.6000 | 0.6319 | 0.0319 |
| | Security | 0.3333 | 0.2528 | -0.0805 |
| | ToolsUtilities | 0.6667 | 0.8215 | 0.1548 |
| Education | Culture | 0.5357 | 0.6003 | 0.0646 |
| | FamilyKids | 0.6333 | 0.5372 | -0.0961 |
| | Language | 0.4667 | 0.5687 | 0.1020 |
| | News | 0.4333 | 0.4424 | 0.0091 |
| | ReferenceBooks | 0.5714 | 0.5687 | -0.0027 |
| | Science | 0.7000 | 0.5687 | -0.1313 |
| | Graphics | 0.3000 | 0.6635 | 0.3635 |
| Creation | Music | 0.4667 | 0.5687 | 0.1020 |
| | Notebook | 0.6897 | 0.6951 | 0.0054 |
| | Photo | 0.5333 | 0.4739 | -0.0594 |
| | Schedule | 0.5333 | 0.6319 | 0.0986 |
| | Video | 0.4667 | 0.4739 | 0.0072 |
| | ActionGame | 0.3667 | 0.4107 | 0.0440 |
| Game | BoardGame | 0.5000 | 0.6319 | 0.1319 |
| | ManagementGame | 0.4000 | 0.5055 | 0.1055 |
| | PuzzleGame | 0.7000 | 0.6635 | -0.0365 |
| | RacingGame | 0.7000 | 0.2528 | -0.4472 |
| | RolePlayingGame | 0.4333 | 0.6003 | 0.1670 |
| | ShooterGame | 0.4000 | 0.4424 | 0.0424 |
| | SimulationGame | 0.4667 | 0.4424 | -0.0243 |
| | SportGame | 0.6333 | 0.7267 | 0.0934 |
| | StrategyGame | 0.2333 | 0.3159 | 0.0826 |
| | All | 0.5022 | 0.5600 | 0.0578 |

Table 7: ChatDev vs GPT-Engineer on completeness.

| Primary Category | Secondary Category | Completeness | | |
|------------------|---------------------|--------------|---------|----------|
| | | MetaGPT | ChatDev | Δ |
| Life | Entertainment | 0.3900 | 0.6319 | 0.2419 |
| | HealthFitness | 0.3567 | 0.7267 | 0.3700 |
| | Medical | 0.4567 | 0.6951 | 0.2384 |
| | Personalisation | 0.6038 | 0.3791 | -0.2247 |
| | RestaurantsDelivery | 0.5003 | 0.4739 | -0.0264 |
| | Shopping | 0.5003 | 0.4739 | -0.0264 |
| | SocialNetwork | 0.5114 | 0.5687 | 0.0573 |
| | Sports | 0.4900 | 0.6003 | 0.1103 |
| | Transportation | 0.4314 | 0.4424 | 0.0110 |
| | Travel | 0.4233 | 0.4739 | 0.0506 |
| Work | Budgeting | 0.3900 | 0.7899 | 0.3999 |
| | Business | 0.5233 | 0.5687 | 0.0454 |
| | Data | 0.6233 | 0.6635 | 0.0402 |
| | Development | 0.7233 | 0.6951 | -0.0282 |
| | Finance | 0.4900 | 0.7267 | 0.2367 |
| | Office | 0.5567 | 0.6319 | 0.0752 |
| | Security | 0.4043 | 0.2528 | -0.1515 |
| Education | ToolsUtilities | 0.7233 | 0.8215 | 0.0982 |
| | Culture | 0.4314 | 0.6003 | 0.1689 |
| | FamilyKids | 0.2567 | 0.5372 | 0.2805 |
| | Language | 0.5233 | 0.5687 | 0.0454 |
| | News | 0.4233 | 0.4424 | 0.0191 |
| | ReferenceBooks | 0.4233 | 0.5687 | 0.1454 |
| | Science | 0.4567 | 0.5687 | 0.1120 |
| Creation | Graphics | 0.4900 | 0.6635 | 0.1735 |
| | Music | 0.4233 | 0.5687 | 0.1454 |
| | Notebook | 0.5233 | 0.6951 | 0.1718 |
| | Photo | 0.4567 | 0.4739 | 0.0172 |
| | Schedule | 0.4757 | 0.6319 | 0.1562 |
| | Video | 0.5900 | 0.4739 | -0.1161 |
| | ActionGame | 0.4567 | 0.4107 | -0.0460 |
| Game | BoardGame | 0.6038 | 0.6319 | 0.0281 |
| | ManagementGame | 0.5233 | 0.5055 | -0.0178 |
| | PuzzleGame | 0.4233 | 0.6635 | 0.2402 |
| | RacingGame | 0.2900 | 0.2528 | -0.0372 |
| | RolePlayingGame | 0.6567 | 0.6003 | -0.0564 |
| | ShooterGame | 0.3900 | 0.4424 | 0.0524 |
| | SimulationGame | 0.4567 | 0.4424 | -0.0143 |
| | SportGame | 0.5233 | 0.7267 | 0.2034 |
| | StrategyGame | 0.4400 | 0.3159 | -0.1241 |
| | All | 0.4834 | 0.5600 | 0.0766 |

Table 8: ChatDev vs MetaGPT on completeness.

| Primary Category | Secondary Category | Executability | | |
|------------------|---------------------|---------------|---------|----------|
| | | GPT-Engineer | ChatDev | Δ |
| Life | Entertainment | 0.2000 | 0.8686 | 0.6686 |
| | HealthFitness | 0.1935 | 0.9440 | 0.7505 |
| | Medical | 0.3548 | 0.9063 | 0.5515 |
| | Personalisation | 0.5313 | 0.8686 | 0.3373 |
| | RestaurantsDelivery | 0.3667 | 0.8307 | 0.4640 |
| | Shopping | 0.4000 | 0.8307 | 0.4307 |
| | SocialNetwork | 0.6333 | 1.0196 | 0.3863 |
| | Sports | 0.3667 | 1.0196 | 0.6529 |
| | Transportation | 0.1667 | 0.9063 | 0.7396 |
| | Travel | 0.4333 | 1.0196 | 0.5863 |
| Work | Budgeting | 0.3667 | 0.9818 | 0.6151 |
| | Business | 0.3571 | 0.9818 | 0.6247 |
| | Data | 0.4667 | 0.9867 | 0.5200 |
| | Development | 0.4333 | 0.9063 | 0.4730 |
| | Finance | 0.5667 | 0.9440 | 0.3773 |
| | Office | 0.2069 | 0.9818 | 0.7749 |
| | Security | 0.1667 | 0.9440 | 0.7773 |
| | ToolsUtilities | 0.5667 | 0.6797 | 0.1130 |
| Education | Culture | 0.4000 | 0.8307 | 0.4307 |
| | FamilyKids | 0.5556 | 0.9063 | 0.3507 |
| | Language | 0.2667 | 0.9818 | 0.7151 |
| | News | 0.3226 | 0.8307 | 0.5081 |
| | ReferenceBooks | 0.2667 | 0.9818 | 0.7151 |
| | Science | 0.2333 | 0.8307 | 0.5974 |
| | Graphics | 0.2333 | 0.8307 | 0.5974 |
| Creation | Music | 0.4286 | 0.6797 | 0.2511 |
| | Notebook | 0.2000 | 0.9818 | 0.7818 |
| | Photo | 0.2000 | 0.9063 | 0.7063 |
| | Schedule | 0.6207 | 0.8686 | 0.2479 |
| | Video | 0.3592 | 0.5287 | 0.1695 |
| | ActionGame | 0.2667 | 0.5664 | 0.2997 |
| | BoardGame | 0.1724 | 0.9440 | 0.7716 |
| Game | ManagementGame | 0.5333 | 0.8686 | 0.3353 |
| | PuzzleGame | 0.5313 | 0.8307 | 0.2994 |
| | RacingGame | 0.3571 | 0.6042 | 0.2471 |
| | RolePlayingGame | 0.3333 | 1.0573 | 0.7240 |
| | ShooterGame | 0.3750 | 0.6042 | 0.2292 |
| | SimulationGame | 0.3000 | 0.9440 | 0.6440 |
| | SportGame | 0.2667 | 0.9818 | 0.7151 |
| | StrategyGame | 0.3333 | 1.0196 | 0.6863 |
| | All | 0.3583 | 0.8800 | 0.5216 |

Table 9: ChatDev vs GPT-Engineer on executability.

| Primary Category | Secondary Category | Executability | | |
|------------------|---------------------|---------------|---------|----------|
| | | MetaGPT | ChatDev | Δ |
| Life | Entertainment | 0.3214 | 0.8686 | 0.5472 |
| | HealthFitness | 0.5333 | 0.9440 | 0.4107 |
| | Medical | 0.4483 | 0.9063 | 0.4580 |
| | Personalisation | 0.5185 | 0.8686 | 0.3501 |
| | RestaurantsDelivery | 0.6786 | 0.8307 | 0.1521 |
| | Shopping | 0.4483 | 0.8307 | 0.3824 |
| | SocialNetwork | 0.6333 | 1.0196 | 0.3863 |
| | Sports | 0.4333 | 1.0196 | 0.5863 |
| | Transportation | 0.5000 | 0.9063 | 0.4063 |
| | Travel | 0.3125 | 1.0196 | 0.7071 |
| Work | Budgeting | 0.4667 | 0.9818 | 0.5151 |
| | Business | 0.3333 | 0.9818 | 0.6485 |
| | Data | 0.3667 | 0.9867 | 0.6200 |
| | Development | 0.3667 | 0.9063 | 0.5396 |
| | Finance | 0.6667 | 0.9440 | 0.2773 |
| | Office | 0.4333 | 0.9818 | 0.5485 |
| | Security | 0.3667 | 0.9440 | 0.5773 |
| Education | ToolsUtilities | 0.4000 | 0.6797 | 0.2797 |
| | Culture | 0.4000 | 0.8307 | 0.4307 |
| | FamilyKids | 0.2667 | 0.9063 | 0.6396 |
| | Language | 0.6667 | 0.9818 | 0.3151 |
| | News | 0.3000 | 0.8307 | 0.5307 |
| | ReferenceBooks | 0.4667 | 0.9818 | 0.5151 |
| | Science | 0.3448 | 0.8307 | 0.4859 |
| Creation | Graphics | 0.3103 | 0.8307 | 0.5204 |
| | Music | 0.4839 | 0.6797 | 0.1958 |
| | Notebook | 0.4667 | 0.9818 | 0.5151 |
| | Photo | 0.2581 | 0.9063 | 0.6482 |
| | Schedule | 0.5000 | 0.8686 | 0.3686 |
| | Video | 0.2667 | 0.5287 | 0.2620 |
| | ActionGame | 0.3333 | 0.5664 | 0.2331 |
| Game | BoardGame | 0.5000 | 0.9440 | 0.4440 |
| | ManagementGame | 0.3793 | 0.8686 | 0.4893 |
| | PuzzleGame | 0.3667 | 0.8307 | 0.4640 |
| | RacingGame | 0.2333 | 0.6042 | 0.3709 |
| | RolePlayingGame | 0.4000 | 1.0573 | 0.6573 |
| | ShooterGame | 0.3000 | 0.6042 | 0.3042 |
| | SimulationGame | 0.4194 | 0.9440 | 0.5246 |
| | SportGame | 0.3333 | 0.9818 | 0.6485 |
| | StrategyGame | 0.3571 | 1.0196 | 0.6625 |
| | All | 0.4145 | 0.8800 | 0.4655 |

Table 10: ChatDev vs MetaGPT on executability.

| Primary Category | Secondary Category | Consistency | | |
|------------------|---------------------|--------------|---------|----------|
| | | GPT-Engineer | ChatDev | Δ |
| Life | Entertainment | 0.7911 | 0.8038 | 0.0127 |
| | HealthFitness | 0.7930 | 0.8068 | 0.0138 |
| | Medical | 0.8009 | 0.8012 | 0.0003 |
| | Personalisation | 0.7868 | 0.7926 | 0.0058 |
| | RestaurantsDelivery | 0.7615 | 0.7842 | 0.0227 |
| | Shopping | 0.7888 | 0.7963 | 0.0075 |
| | SocialNetwork | 0.7633 | 0.7804 | 0.0171 |
| | Sports | 0.7795 | 0.8044 | 0.0249 |
| | Transportation | 0.7760 | 0.8059 | 0.0299 |
| | Travel | 0.7901 | 0.7993 | 0.0092 |
| Work | Budgeting | 0.7943 | 0.8153 | 0.0210 |
| | Business | 0.8030 | 0.8044 | 0.0014 |
| | Data | 0.8046 | 0.8151 | 0.0105 |
| | Development | 0.8257 | 0.8247 | -0.0010 |
| | Finance | 0.7848 | 0.8209 | 0.0361 |
| | Office | 0.7990 | 0.7969 | -0.0021 |
| | Security | 0.7723 | 0.7924 | 0.0201 |
| Education | ToolsUtilities | 0.8079 | 0.8184 | 0.0105 |
| | Culture | 0.7704 | 0.7806 | 0.0102 |
| | FamilyKids | 0.7816 | 0.7895 | 0.0079 |
| | Language | 0.7962 | 0.8079 | 0.0117 |
| | News | 0.7981 | 0.7976 | -0.0005 |
| | ReferenceBooks | 0.7806 | 0.8051 | 0.0245 |
| | Science | 0.7971 | 0.7915 | -0.0056 |
| Creation | Graphics | 0.7958 | 0.8161 | 0.0203 |
| | Music | 0.7936 | 0.8246 | 0.0310 |
| | Notebook | 0.7909 | 0.8077 | 0.0168 |
| | Photo | 0.7978 | 0.8059 | 0.0081 |
| | Schedule | 0.8054 | 0.8109 | 0.0055 |
| | Video | 0.7982 | 0.8058 | 0.0076 |
| | ActionGame | 0.7675 | 0.7814 | 0.0139 |
| Game | BoardGame | 0.8123 | 0.8097 | -0.0026 |
| | ManagementGame | 0.7854 | 0.7945 | 0.0091 |
| | PuzzleGame | 0.7876 | 0.8021 | 0.0145 |
| | RacingGame | 0.7364 | 0.7852 | 0.0488 |
| | RolePlayingGame | 0.8099 | 0.8126 | 0.0027 |
| | ShooterGame | 0.7562 | 0.7792 | 0.0230 |
| | SimulationGame | 0.7905 | 0.8054 | 0.0149 |
| | SportGame | 0.7891 | 0.8133 | 0.0242 |
| | StrategyGame | 0.7852 | 0.7959 | 0.0107 |
| | All | 0.7887 | 0.8021 | 0.0134 |

Table 11: ChatDev vs GPT-Engineer on consistency.

| Primary Category | Secondary Category | Consistency | | |
|------------------|---------------------|-------------|---------|----------|
| | | MetaGPT | ChatDev | Δ |
| Life | Entertainment | 0.7695 | 0.8038 | 0.0343 |
| | HealthFitness | 0.7500 | 0.8068 | 0.0568 |
| | Medical | 0.7573 | 0.8012 | 0.0439 |
| | Personalisation | 0.7424 | 0.7926 | 0.0502 |
| | RestaurantsDelivery | 0.7501 | 0.7842 | 0.0341 |
| | Shopping | 0.7433 | 0.7963 | 0.0530 |
| | SocialNetwork | 0.7159 | 0.7804 | 0.0645 |
| | Sports | 0.7579 | 0.8044 | 0.0465 |
| | Transportation | 0.7638 | 0.8059 | 0.0421 |
| | Travel | 0.7584 | 0.7993 | 0.0409 |
| Work | Budgeting | 0.7433 | 0.8153 | 0.0720 |
| | Business | 0.7392 | 0.8044 | 0.0652 |
| | Data | 0.7881 | 0.8151 | 0.0270 |
| | Development | 0.7841 | 0.8247 | 0.0406 |
| | Finance | 0.7609 | 0.8209 | 0.0600 |
| | Office | 0.7438 | 0.7969 | 0.0531 |
| | Security | 0.7548 | 0.7924 | 0.0376 |
| | ToolsUtilities | 0.7930 | 0.8184 | 0.0254 |
| Education | Culture | 0.7517 | 0.7806 | 0.0289 |
| | FamilyKids | 0.7469 | 0.7895 | 0.0426 |
| | Language | 0.7636 | 0.8079 | 0.0443 |
| | News | 0.7698 | 0.7976 | 0.0278 |
| | ReferenceBooks | 0.7521 | 0.8051 | 0.0530 |
| | Science | 0.7594 | 0.7915 | 0.0321 |
| | Graphics | 0.7887 | 0.8161 | 0.0274 |
| Creation | Music | 0.7962 | 0.8246 | 0.0284 |
| | Notebook | 0.7615 | 0.8077 | 0.0462 |
| | Photo | 0.7888 | 0.8059 | 0.0171 |
| | Schedule | 0.7329 | 0.8109 | 0.0780 |
| | Video | 0.7571 | 0.8058 | 0.0487 |
| | ActionGame | 0.7493 | 0.7814 | 0.0321 |
| | BoardGame | 0.7813 | 0.8097 | 0.0284 |
| Game | ManagementGame | 0.7538 | 0.7945 | 0.0407 |
| | PuzzleGame | 0.7679 | 0.8021 | 0.0342 |
| | RacingGame | 0.7483 | 0.7852 | 0.0369 |
| | RolePlayingGame | 0.7724 | 0.8126 | 0.0402 |
| | ShooterGame | 0.7542 | 0.7792 | 0.0250 |
| | SimulationGame | 0.7711 | 0.8054 | 0.0343 |
| | SportGame | 0.7633 | 0.8133 | 0.0500 |
| | StrategyGame | 0.7573 | 0.7959 | 0.0386 |
| | All | 0.7601 | 0.8021 | 0.0421 |

Table 12: ChatDev vs MetaGPT on consistency.

| Primary Category | Secondary Category | Quality | | |
|------------------|---------------------|--------------|---------|----------|
| | | GPT-Engineer | ChatDev | Δ |
| Life | Entertainment | 0.0622 | 0.4412 | 0.3790 |
| | HealthFitness | 0.0716 | 0.5535 | 0.4819 |
| | Medical | 0.1989 | 0.5047 | 0.3058 |
| | Personalisation | 0.2090 | 0.2610 | 0.0520 |
| | RestaurantsDelivery | 0.1210 | 0.3087 | 0.1877 |
| | Shopping | 0.1262 | 0.3135 | 0.1873 |
| | SocialNetwork | 0.1450 | 0.4525 | 0.3075 |
| | Sports | 0.1715 | 0.4923 | 0.3208 |
| | Transportation | 0.0561 | 0.3231 | 0.2670 |
| | Travel | 0.1369 | 0.3862 | 0.2493 |
| Work | Budgeting | 0.1942 | 0.6323 | 0.4381 |
| | Business | 0.1529 | 0.4492 | 0.2963 |
| | Data | 0.1502 | 0.5337 | 0.3835 |
| | Development | 0.2028 | 0.5195 | 0.3167 |
| | Finance | 0.2372 | 0.5632 | 0.3260 |
| | Office | 0.0992 | 0.4944 | 0.3953 |
| | Security | 0.0429 | 0.1891 | 0.1462 |
| | ToolsUtilities | 0.3052 | 0.4570 | 0.1518 |
| Education | Culture | 0.1651 | 0.3893 | 0.2242 |
| | FamilyKids | 0.2750 | 0.3843 | 0.1093 |
| | Language | 0.0991 | 0.4511 | 0.3520 |
| | News | 0.1116 | 0.2931 | 0.1815 |
| | ReferenceBooks | 0.1190 | 0.4496 | 0.3306 |
| | Science | 0.1302 | 0.3739 | 0.2438 |
| | Graphics | 0.0557 | 0.4498 | 0.3941 |
| Creation | Music | 0.1587 | 0.3187 | 0.1600 |
| | Notebook | 0.1091 | 0.5512 | 0.4421 |
| | Photo | 0.0851 | 0.3462 | 0.2611 |
| | Schedule | 0.2666 | 0.4451 | 0.1785 |
| | Video | 0.1338 | 0.2019 | 0.0681 |
| | ActionGame | 0.0751 | 0.1818 | 0.1067 |
| Game | BoardGame | 0.0700 | 0.4830 | 0.4130 |
| | ManagementGame | 0.1675 | 0.3488 | 0.1813 |
| | PuzzleGame | 0.2929 | 0.4421 | 0.1492 |
| | RacingGame | 0.1841 | 0.1199 | -0.0642 |
| | RolePlayingGame | 0.1170 | 0.5158 | 0.3988 |
| | ShooterGame | 0.1134 | 0.2083 | 0.0948 |
| | SimulationGame | 0.1107 | 0.3363 | 0.2257 |
| | SportGame | 0.1333 | 0.5803 | 0.4471 |
| | StrategyGame | 0.0611 | 0.2564 | 0.1953 |
| | All | 0.1419 | 0.3953 | 0.2571 |

Table 13: ChatDev vs GPT-Engineer on comprehensive quality.

| Primary Category | Secondary Category | Quality | | |
|------------------|---------------------|---------|---------|----------|
| | | MetaGPT | ChatDev | Δ |
| Life | Entertainment | 0.0965 | 0.4412 | 0.3447 |
| | HealthFitness | 0.1427 | 0.5535 | 0.4108 |
| | Medical | 0.1550 | 0.5047 | 0.3497 |
| | Personalisation | 0.2324 | 0.2610 | 0.0286 |
| | RestaurantsDelivery | 0.2547 | 0.3087 | 0.0541 |
| | Shopping | 0.1667 | 0.3135 | 0.1468 |
| | SocialNetwork | 0.2319 | 0.4525 | 0.2207 |
| | Sports | 0.1609 | 0.4923 | 0.3314 |
| | Transportation | 0.1648 | 0.3231 | 0.1583 |
| | Travel | 0.1003 | 0.3862 | 0.2859 |
| Work | Budgeting | 0.1353 | 0.6323 | 0.4970 |
| | Business | 0.1289 | 0.4492 | 0.3202 |
| | Data | 0.1801 | 0.5337 | 0.3535 |
| | Development | 0.2080 | 0.5195 | 0.3115 |
| | Finance | 0.2486 | 0.5632 | 0.3146 |
| | Office | 0.1794 | 0.4944 | 0.3150 |
| | Security | 0.1119 | 0.1891 | 0.0772 |
| Education | ToolsUtilities | 0.2294 | 0.4570 | 0.2276 |
| | Culture | 0.1297 | 0.3893 | 0.2596 |
| | FamilyKids | 0.0511 | 0.3843 | 0.3332 |
| | Language | 0.2664 | 0.4511 | 0.1847 |
| | News | 0.0978 | 0.2931 | 0.1953 |
| | ReferenceBooks | 0.1486 | 0.4496 | 0.3010 |
| | Science | 0.1196 | 0.3739 | 0.2543 |
| Creation | Graphics | 0.1199 | 0.4498 | 0.3299 |
| | Music | 0.1631 | 0.3187 | 0.1557 |
| | Notebook | 0.1860 | 0.5512 | 0.3652 |
| | Photo | 0.0930 | 0.3462 | 0.2532 |
| | Schedule | 0.1743 | 0.4451 | 0.2708 |
| | Video | 0.1191 | 0.2019 | 0.0828 |
| | ActionGame | 0.1141 | 0.1818 | 0.0677 |
| Game | BoardGame | 0.2359 | 0.4830 | 0.2472 |
| | ManagementGame | 0.1496 | 0.3488 | 0.1992 |
| | PuzzleGame | 0.1192 | 0.4421 | 0.3229 |
| | RacingGame | 0.0506 | 0.1199 | 0.0693 |
| | RolePlayingGame | 0.2029 | 0.5158 | 0.3129 |
| | ShooterGame | 0.0882 | 0.2083 | 0.1200 |
| | SimulationGame | 0.1477 | 0.3363 | 0.1886 |
| | SportGame | 0.1331 | 0.5803 | 0.4472 |
| | StrategyGame | 0.1190 | 0.2564 | 0.1374 |
| | All | 0.1523 | 0.3953 | 0.2461 |

Table 14: ChatDev vs MetaGPT on comprehensive quality.

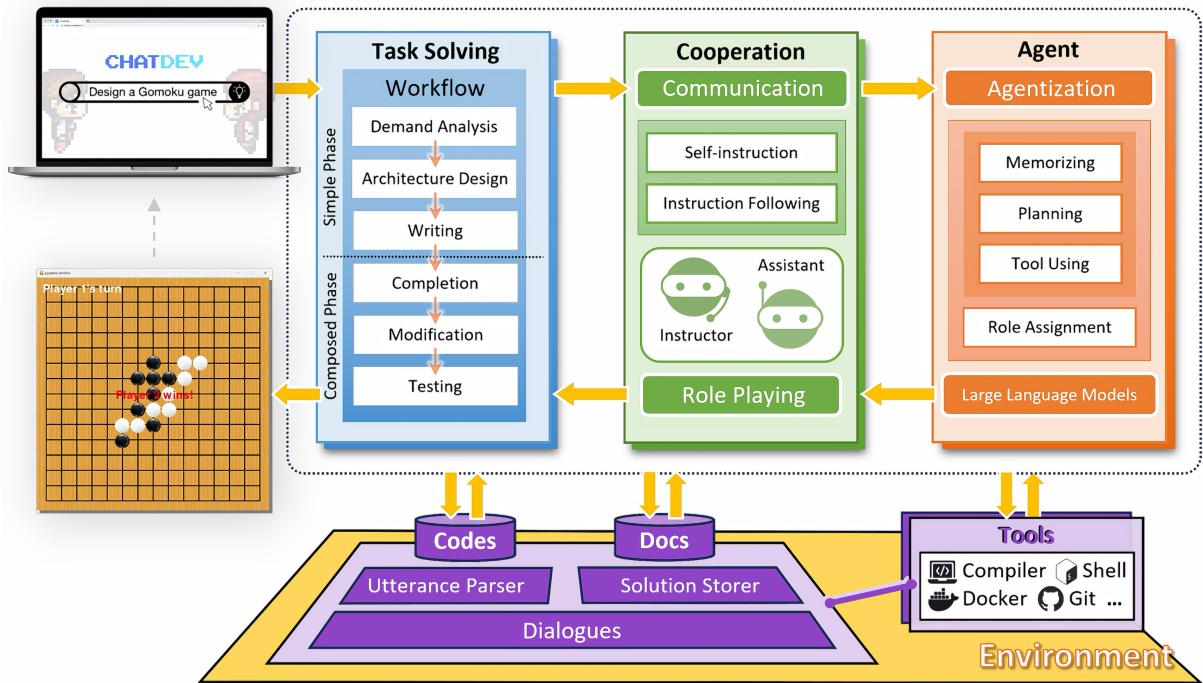


Figure 22: The system architecture and main components of ChatDev.

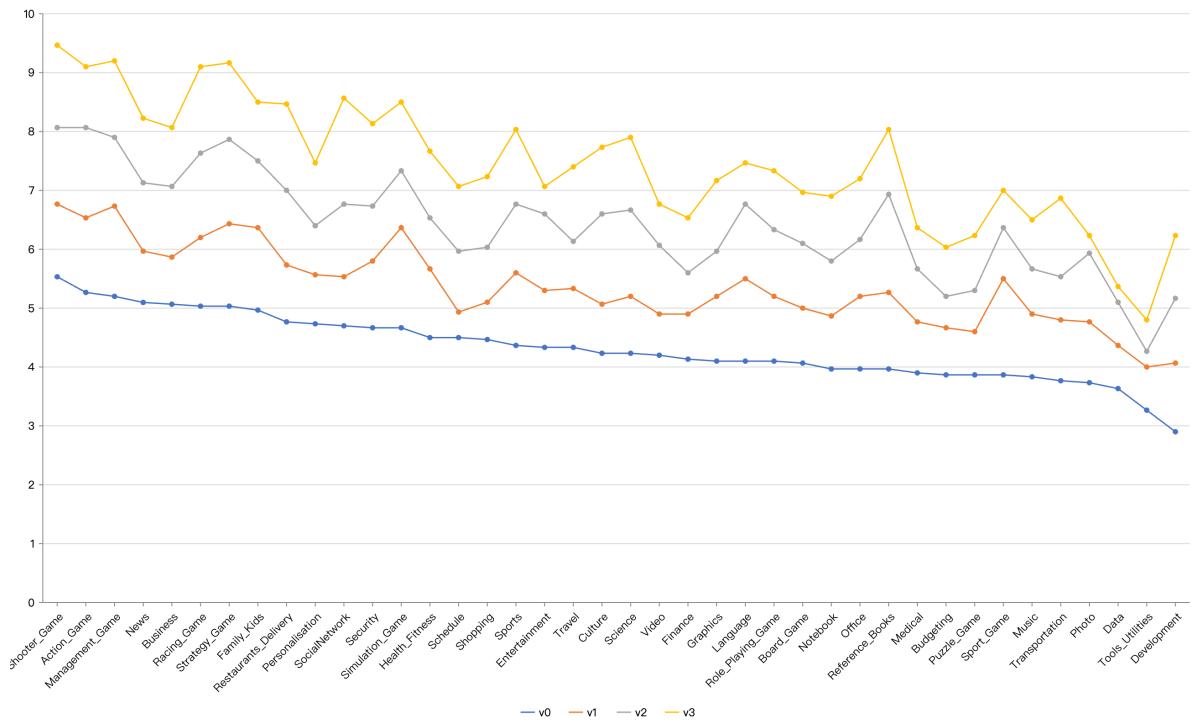


Figure 23: The number of code files across different versions.

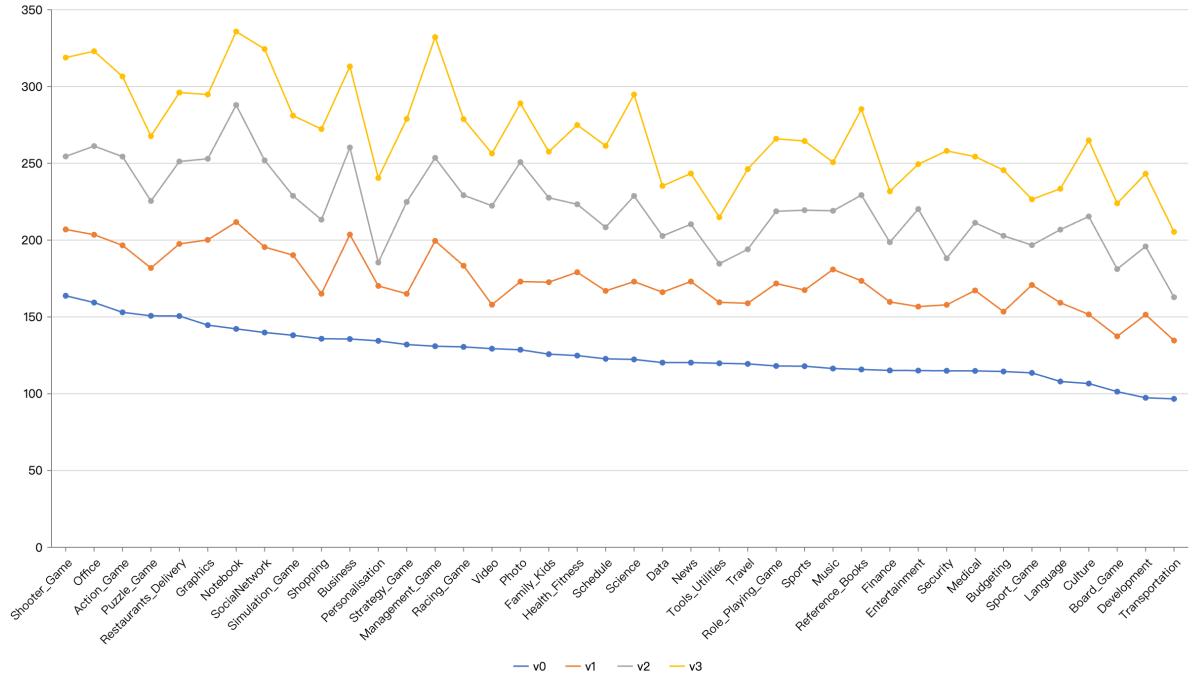


Figure 24: The number of code lines across different versions.

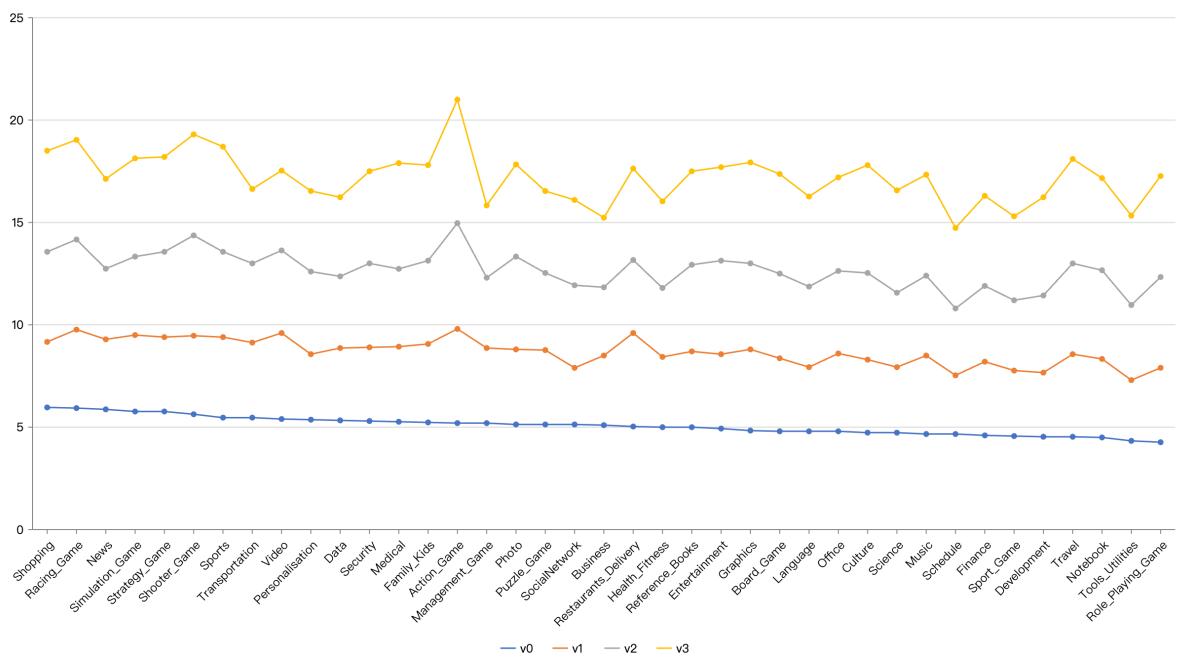


Figure 25: The number of code updates across different versions.

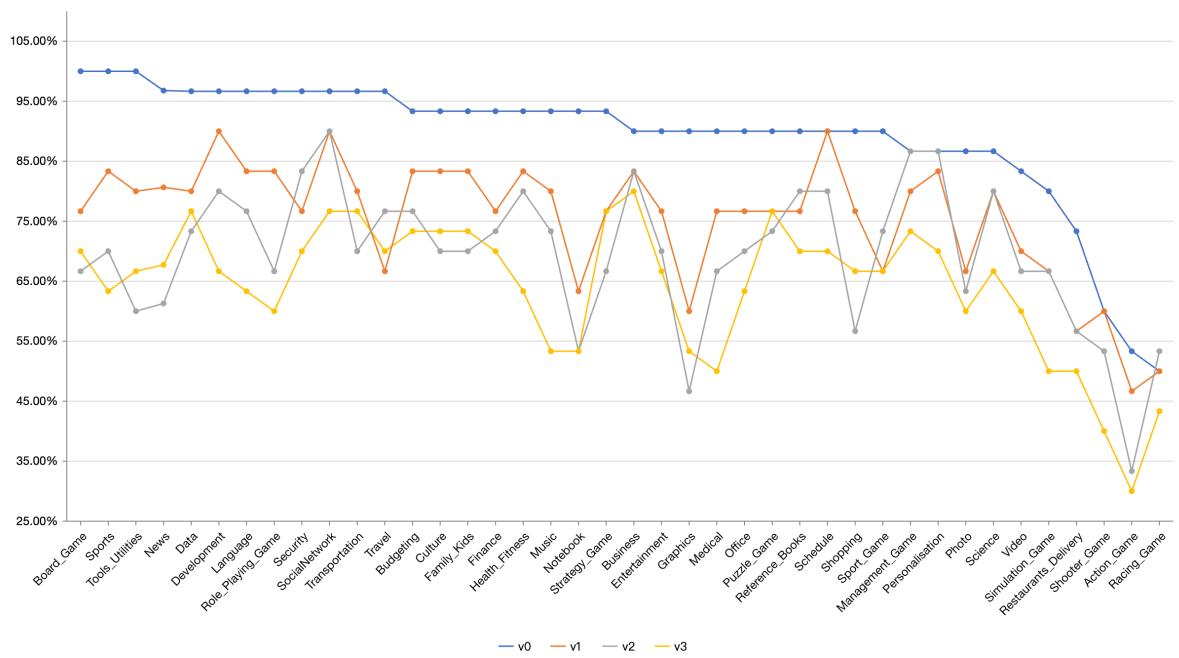


Figure 26: The executability metric across different versions.

Demands Classification

Here's a summary of the software demands from 13 perspectives, each given with the name and definition of the perspective:

1. Target Users: Target Users denotes the characteristics of the main users for this software, including:
 - Individuals: (finance, health, productivity, entertainment, personal development)
 - Businesses: (feedback analysis, sales tracking, security, productivity)
 - Developers: (code coverage, data analysis)
 - Families: (art studio, scrapbooking, chore challenge, education, entertainment)
 - Social Groups: (group chat, live streaming, cultural connections)
2. Platforms: On which platform will this software be running, for example,
 - Desktop Applications: (directory size analyzer, paintpal, code coverage heatmap, etc.)
 - Mobile Apps: (food delivery, travel companion, symptom tracker, livestreaming, finance tracker, etc.)
 - Web Applications: (product feedback analyzer, social livestream, smart shopper)
 - Command-Line Tools: (console file renamer)
 - Cross-Platform Compatibility: (finance, symptom tracking, note synchronization, task planning)
3. Performance: The demands mention the importance of optimizing software performance and robustness. This includes implementing efficient algorithms, data processing and retrieval, minimizing latency, caching, parallel processing, error handling, and user control to improve performance and ensure reliable operation. It is common in data analysis, gaming, and live streaming software. The software should prioritize performance optimization to ensure smooth operation.
4. Customization: Customization demands mention the need for customization options, such as customizable user profiles, customizable themes, filters, and settings. Customization allows users to tailor the software to their preferences and enhance their experience, learning paths, and the ability to customize settings and features.
5. Recommendation: The applications should implement advanced algorithms, such as machine learning, natural language processing, and optimization algorithms, to enhance functionality improve user experience, and offer personalized experiences based on user preferences and behavior.
6. Integration: The Integration demands mention the integration of software applications with external sources, such as online databases, APIs, wearable devices, social media platforms, financial institutions, external databases, and online news sources. This integration allows for real-time data retrieval, synchronization, and enhanced functionality.
7. UI/UX: (User Interface and User Experience) demands a focus on the usage process of users. Software with good User Experience should perform well in the following aspects:
 - Intuitiveness: Easy to learn and use
 - Engagement: Keeps users interested and motivated
 - Accessibility: Caters to diverse needs and abilitiesThe User Interface demands emphasize the need for user-friendly interfaces with clean designs, intuitive navigation, visually appealing graphics, and responsive layouts. The interfaces should prioritize simplicity, clarity, and ease of use to enhance the user experience.
8. Data Management: The demands highlight the importance of data management, including storing and retrieving data, syncing across devices, handling large datasets, and ensuring data privacy and security. This includes features like saving/loading preferences, syncing progress, and implementing encryption and authentication mechanisms. Specifically, it includes:
 - Data Storage: Secure and efficient storage of user data
 - Data Privacy: Protection of user information
 - Data Analysis: Ability to extract insights from data
 - Data Integration: Ability to connect with other systems and sources
 - Data Import/Export: (data analysis, feedback analysis)
 - Data Visualization: (heatmaps, charts, graphs)
 - Data Interaction: The software should be able to interact with online data sources, local files, or databases to fetch, store, and analyze data. This includes simulating data for testing purposes and ensuring data accuracy and reliability.
9. Scalability: Including:
 - Ability to handle growth: Accommodate increasing numbers of users and data
 - Adaptability to change: Evolve with changing needs and technologies
10. Maintenance: Including:
 - Bug fixes and updates: Regular maintenance and improvement
 - User support: Assistance for troubleshooting and questions
11. Communication/Collaboration: The applications should facilitate collaboration and communication among users. This includes features like real-time chat, shared workspaces, and the ability to assign and track tasks. Some demands mention the need for collaboration and communication features, such as real-time comments and reactions during live streams, notifications for upcoming live streams and new followers, collaboration on task management, and sharing of meeting details and updates.
12. Real-Time Updates: The applications should provide real-time updates and notifications to keep users informed. This includes progress tracking, alerts for upcoming events, and real-time feedback. For example, getting real-time updates on travel, news, or symptom tracking.
13. Security and Privacy: The software should prioritize data security and privacy. This includes implementing encryption and authentication mechanisms, ensuring data privacy, and complying with relevant privacy regulations. For example, threat hunting, phishing protection, and malware protection.

Now I will give you a text of software demand. The text is: "{Software Description}"
It is a {Software Category} software named {Software Name} please classify this text into one of the above 13 perspectives. You can classify it from more than one perspective. You must return in the format like [perspective1, perspective2]. The perspective can only be chosen from (Target Users, Platforms, Performance, Customization, Recommendation, Integration, UI/UX, Data Management, Scalability, Maintenance, Communication/Collaboration, Real-Time Updates, Security and Privacy) Remember only return the list of names of perspectives! For example, return [Target Users, Platforms, Performance]

Figure 27: The prompt of demands classification.

Reviews Classification

I will give you a software code review comment.
Here is the comment:
{Comment Content}
Please classify it into one of the following types Method Not Implemented, Modules Not Imported, Missing Comments, Missing Exception Handling, Calling without Correct Arguments, Class Defined Twice, Missing Files, Not Handle Exceptions, Class Not Used, Infinite Loop, Methods Not Called, Missing Code Segments, Missing Initialization, Missing Loop Logics, Not Configure Layout or Functionality, Not Correctly Processing Data, Not Handle Cases, Type Errors, Use Other Layouts
You should only return the name of the type.

Figure 28: The prompt of review classification.

Requirement Implementation

Now you are given a manual of a software and a user requirement of this software. You need to judge how many demand points in the user requirement are realized based on the manual.

Think step by step. You need first summarize the demand points in the user requirement, then judge how many of these points are realized in the manual. Here is the manual:

{Manual Content}

Here is the user requirement:

{User Requirement}

If you summarize x points in the user requirement and y of them are realized in the manual, and for example, the No.k demand point is failed to realize, then you must return in the format like below (This is only an example, the number and content should base on the actual case. the number of ✓ is y and the total number (✓ + ×) is x.)

<RESPONSE>

y/x

[✓] Demand points 1: the content of this demand point

.....

[×] Demand points k: the content of this demand point

.....

[✓] Demand points x: the content of this demand point

Figure 29: The prompt of requirement implementation.