Product flow analysis of chatdev

Similar to hallucinations encountered when using LLMs for natural language querying , directly generating entire software systems using LLMs can result in severe code hallucinations, such as incomplete implementation, missing dependencies, and undiscovered bugs. These hallucinations may stem from the lack of specificity in the task and the absence of cross-examination in decision-making. To address these limitations, we establish a virtual chat-powered software technology company – ChatDev, which comprises recruited agents from diverse social identities, such as chief officers, professional programmers, test engineers, and art designers. When presented with a task, the diverse agents at ChatDev collaborate to develop a required software.

ChatDev employs the widely adopted waterfall model, a prominent software development life cycle model, to divide the software development process into four distinct phases: designing, coding, testing, and documenting.

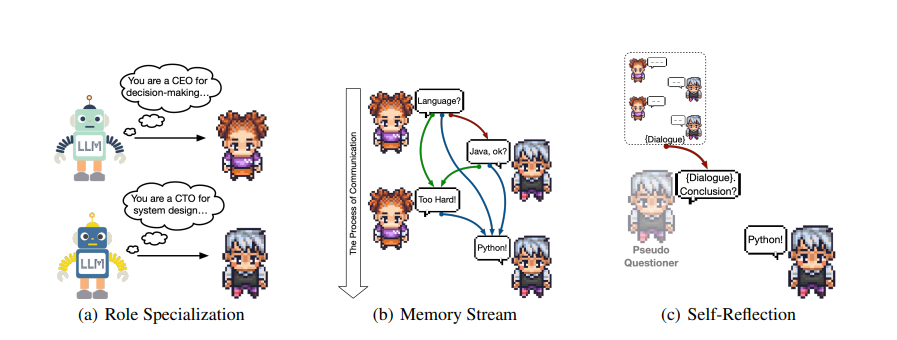
Each of these phases necessitates effective communication among multiple roles, posing challenges in determining the sequence of interactions and identifying the relevant individuals to engage with. To address this, we propose a generalized architecture by breaking down each phase into multiple atomic chats, each with a specific focus on task-oriented role-playing involving two distinct roles.

Through the exchange of instructions and collaboration between the participating agents, the desired output for each chat, which forms a vital component of the target software, is achieved.

Phases

# Designing

In the designing phase, ChatDev receives an initial idea from a human client. This phase involves three predefined roles: CEO (chief executive officer), CPO (chief product officer), and CTO (chief technology officer). The chat chain then breaks down the designing phase into sequential atomic chatting tasks, including decisions regarding the target software’s modality (CEO and CPO) and the programming language (CEO and CTO)



# Coding

The coding phase involves three predefined roles: CTO, programmer, and art designer. The chat chain decomposes the coding phase into sequential atomic chatting tasks, such as generating complete codes (CTO and programmer) and devising a graphical user interface (designer and programmer). Based on the main designs discussed in the previous phase, the CTO instructs the programmer to implement a software system using markdown format. The programmer generates codes in response and extracts the corresponding codes based on markdown format. The designer proposes a user-friendly graphical user interface (GUI) that uses graphical icons for user interaction instead of text-based commands. Then, the designer creates visually appealing graphics using external text-to-image tools , which the programmer incorporates into the GUI design using standard toolkits.

# Testing

Even for human programmers, there is no guarantee that the code they write on the first attempt is always error-free. Rather than discarding incorrect code outright, humans typically analyze and investigate code execution results to identify and rectify implementation errors . In ChatDev, the testing phase involves three roles: programmer, reviewer, and tester. The process consists of sequential atomic chatting tasks, including peer review (programmer and reviewer) and system testing (programmer and tester). Peer review, or static debugging, examines source code to identify potential issues. System testing, a form of dynamic debugging, verifies software execution through tests conducted by the programmer using an interpreter. This testing focuses on evaluating application performance through black-box testing.

# Documenting

After the designing, coding, and testing phases, ChatDev employs four agents (CEO, CPO, CTO, and programmer) to generate software project documentation. Using large language models, we leverage few-shot prompting with in-context examples for document generation. The CTO instructs the programmer to provide configuration instructions for environmental dependencies, resulting in a document like requirements.txt. This document allows users to configure the environment independently. Simultaneously, the CEO communicates requirements and system design to the CPO, who generates a user manual.