



Communicative Agents for Software Development

Chen Qian[♦] Xin Cong[♦] Wei Liu[♦] Cheng Yang[♦] Weize Chen[♦] Yusheng Su[♦]
Yufan Dang[♦] Jiahao Li[♦] Juyuan Xu[♦] Dahai Li[★] Zhiyuan Liu[✉] Maosong Sun[✉]

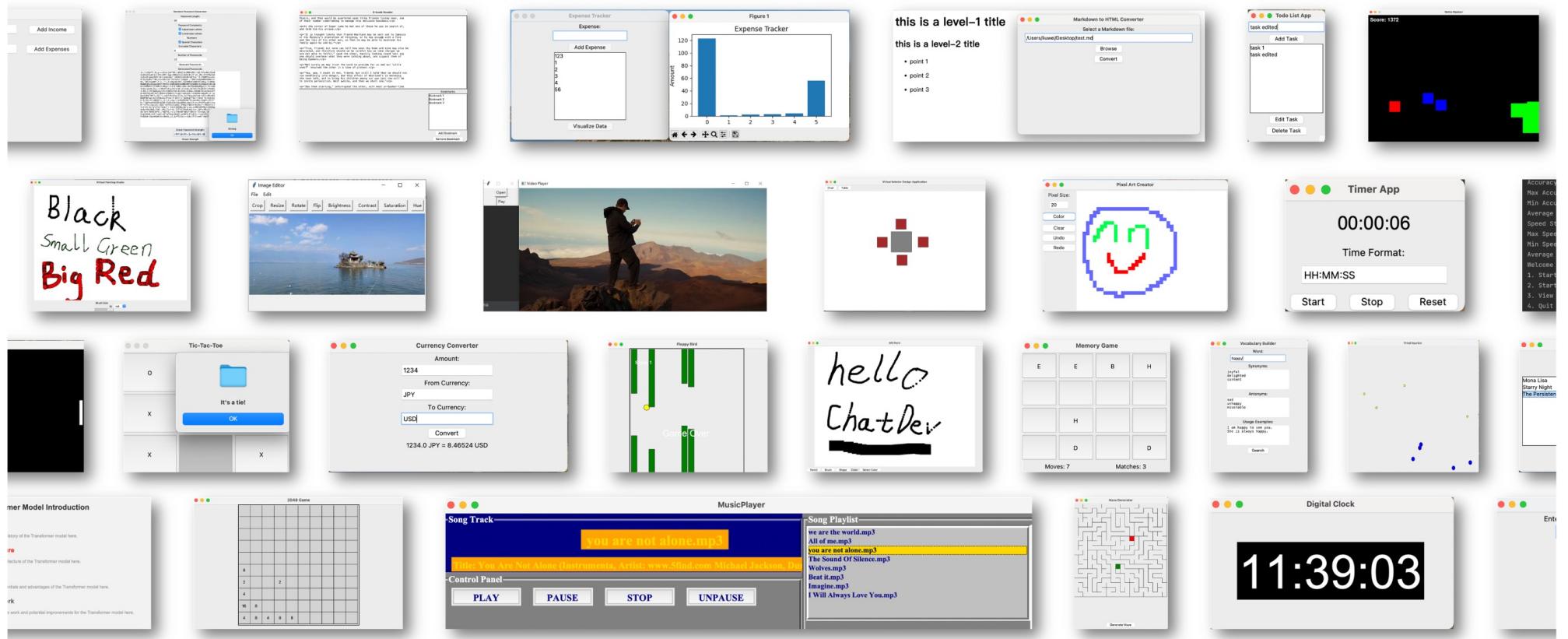
[♦]Tsinghua University [✉]Beijing University of Posts and Telecommunications

[♦]Dalian University of Technology [▲]Brown University [★]Modelbest Inc.

qianc62@gmail.com liuzy@tsinghua.edu.cn sms@tsinghua.edu.cn

Presented by:
John Tan Chong Min

One System. Multiple Programs.



Background: CAMEL - Inception Prompting

Why not let the AI be both the user and the assistant for the LLM interaction?

ChatGPT – only Human user is now an AI!

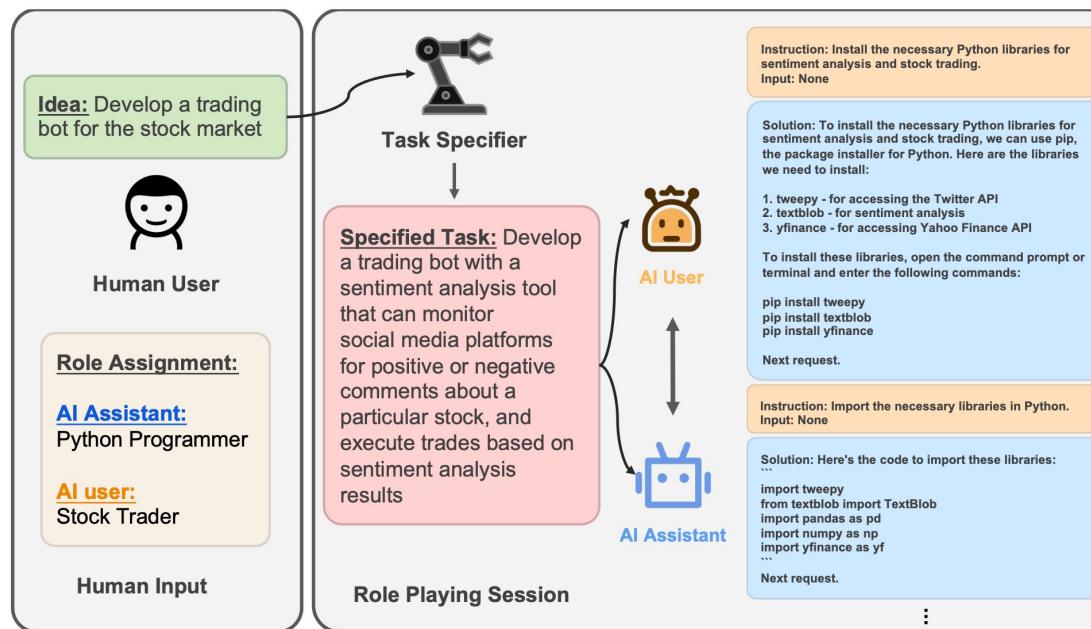


Figure 1: **Role-Playing Framework.** Our role-playing setup starts with the human user having an idea they want to implement, e.g. develop a trading bot for the stock market. The roles involved in this task would be an AI assistant agent who is a python programmer and an AI user agent who is a stock trader. The task is made more specific using our task specifier agent, leading to a well-defined task for the assistant to solve. The AI user and AI assistant collaboratively communicate by chatting with each other in an instruction-following fashion to solve the specified task.

AI User Prompt

For the AI user system prompt $\mathcal{P}_{\mathcal{U}}$, we strive to maintain as much symmetry as possible with respect to the AI assistant system prompt. Apart from the opposite role assignment, the user system prompt differs from the assistant prompt in the following ways:

- You must instruct me based on my expertise and your needs to complete the task ONLY in the following two ways: 1. Instruct with a necessary input: ...; 2. Instruct without any input: This follows the typical data structure of instruction-following, which allows the generated instruction-solution pairs to be easily used for fine-tuning LLMs
- Keep giving me instructions and necessary inputs until you think the task is completed. When the task is completed, you must only reply with a single word <CAMEL_TASK_DONE>. We introduce an end-of-task token, namely, <CAMEL_TASK_DONE>. This token is used once the user believes the task is done. This ensures that the chat is terminated when the user is satisfied. Without doing so, the agents might fall into a chatting loop where they keep on saying “thank you” to each other or “goodbye” indefinitely.

AI Assistant Prompt (Basically ChatGPT)

Prompt Engineering. To delve deeper into the details in Figure 2, we start by chunking the various parts of the AI assistant system prompt $\mathcal{P}_{\mathcal{A}}$ shown below:

- Never forget you are a <ASSISTANT_ROLE> and I am a <USER_ROLE>. This assigns the chosen role to the assistant agent and provides the agent with information about the user's role.
- Never flip roles! Never instruct me! This prevents agents from flipping roles. In some cases, we have observed the assistant and the user switching roles, where the assistant suddenly takes control and instructs the user, and the user follows those instructions.
- You must decline my instruction honestly if you cannot perform the instruction due to physical, moral, legal reasons or your capability and explain the reasons. This prohibits the agent from producing harmful, false, illegal, and misleading information.
- Unless I say the task is completed, you should always start with:
Solution: <YOUR_SOLUTION>. <YOUR_SOLUTION> should be specific, and provide preferable implementations and examples for task-solving.
This encourages the assistant always responds in a consistent format, avoiding any deviation from the structure of the conversation, and preventing vague or incomplete responses, which we refer to as flake responses, such as "I will do something".
- Always end your solution with: Next request. This ensures that the assistant keeps the conversation going by requesting a new instruction to solve.

ChatDev

CAMEL + Guided Process

Chain-of-Thought via Different Agents

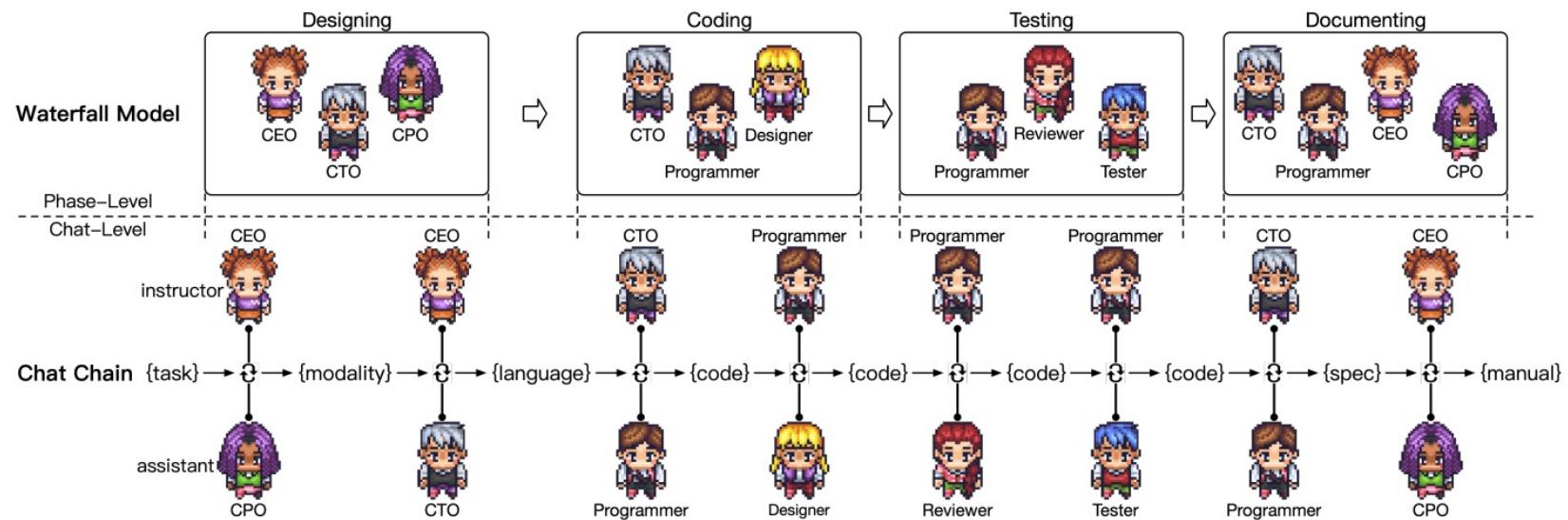
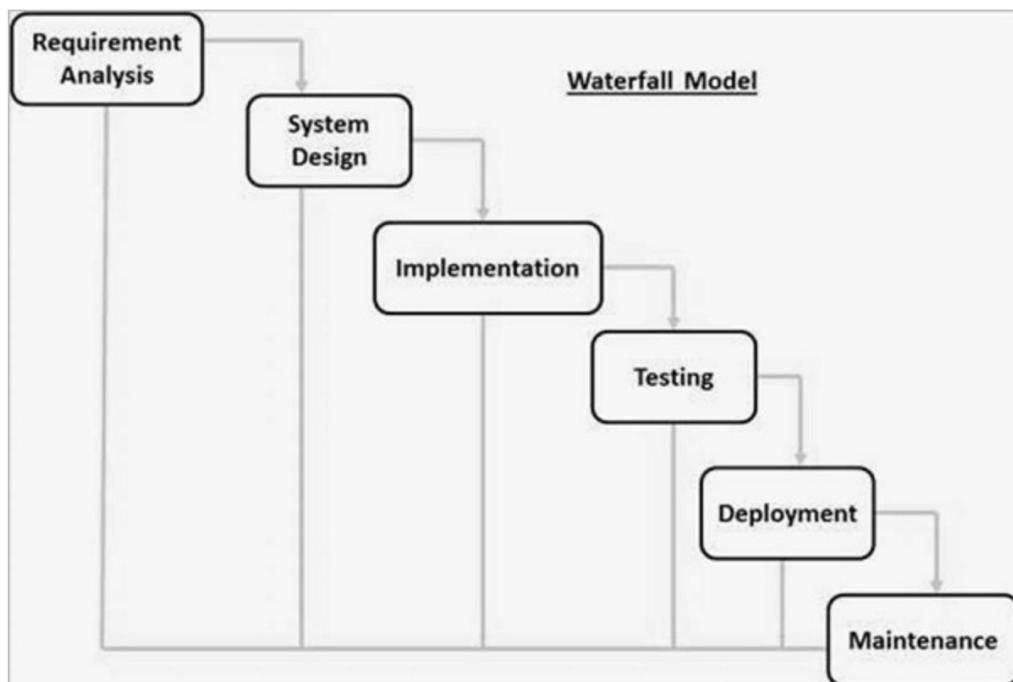


Figure 2: The proposed architecture of CHATDEV consists of phase-level and chat-level components. At the phase level, the waterfall model is used to break down the software development process into four sequential phases. At the chat level, each phase is further divided into atomic chats. These atomic chats involve task-oriented role-playing between two agents, promoting collaborative communication. The communication follows an instruction-following style, where agents interact to accomplish a specific subtask within each chat.

Waterfall Model



- Breakdown of project activities into linear sequential phases, meaning they are passed down onto each other
- Each phase depends on the deliverables of the previous one and corresponds to a specialization of tasks

https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

Waterfall Model

- CHATDEV employs the widely adopted waterfall model, a prominent software development life cycle model
 - **Designing:** Innovative ideas are generated through collaborative brainstorming, and technical design requirements are defined
 - **Coding:** Involves the development and review of source code
 - **Testing:** Integrates all components into a system and utilizes feedback messages from interpreter for debugging
 - **Documenting:** Encompasses the generation of environment specifications and user manuals.

Zero-shot Specialisation

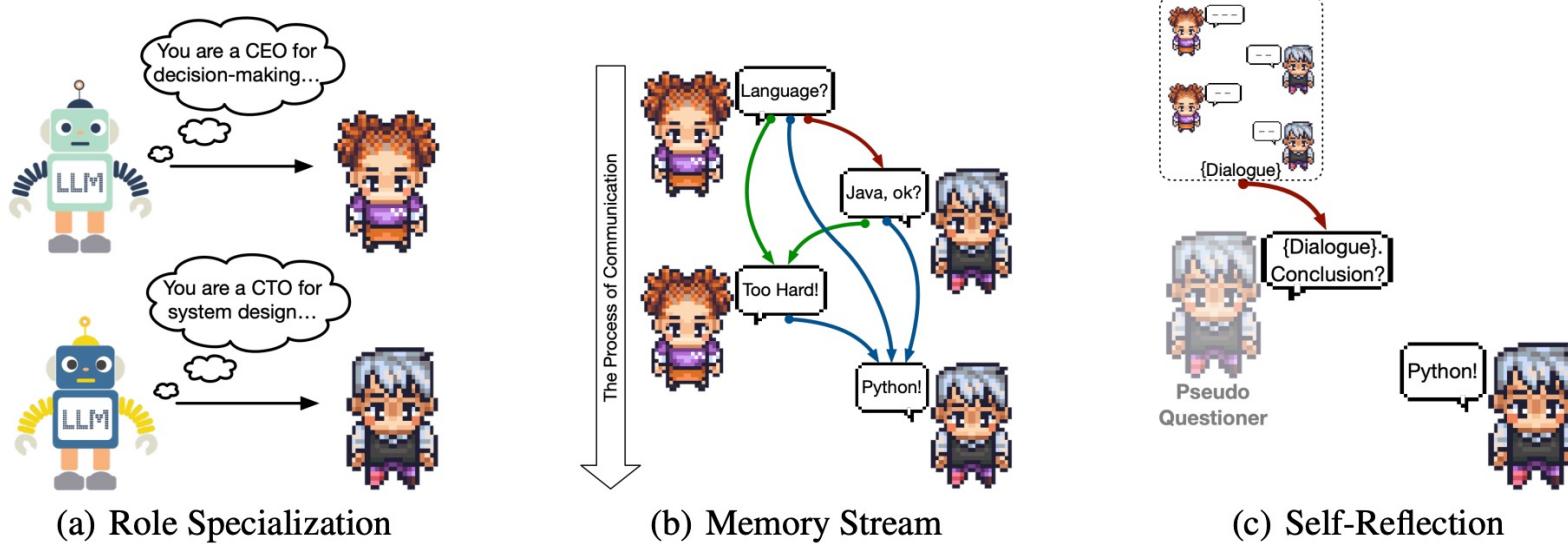
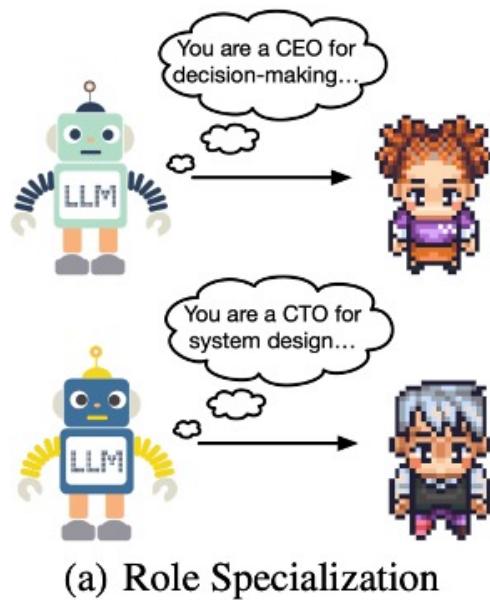


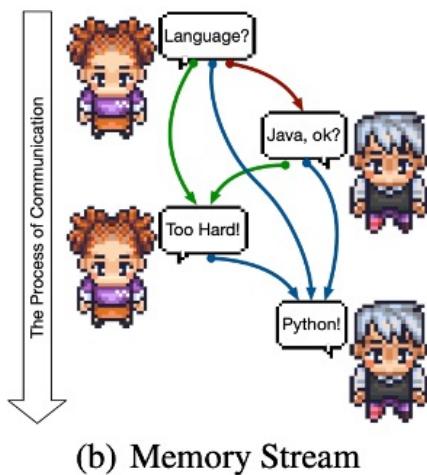
Figure 3: Three key mechanisms utilized in each chat. Role specialization ensures that each agent fulfills their designated functions and contributes effectively to the task-oriented dialogue. The memory stream maintains a comprehensive record of previous dialogues within the chat, enabling agents to make informed decisions. Self-reflection prompts the assistant to reflect on proposed decisions when both parties reach a consensus without triggering predefined termination conditions.

Mechanism 1: Role Specialisation



- Prompt for the role is given before the start of dialogue with all agents
- Serves to condition the agent to output in a certain way
- Every agent can either be a **user** or an **assistant**
 - **User** comes up with intent
 - **Assistant** executes it
 - User and Assistant prompts each other continuously, termed as “Inception Prompting”

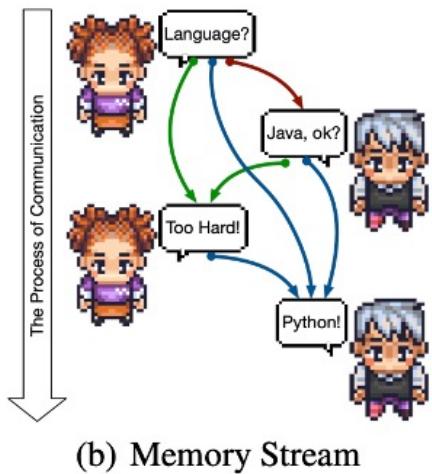
Mechanism 2: Memory Stream



- Stores past messages of user and assistant
- User takes past context and comes up with new instruction for assistant to solve
- Assistant takes past context and user's latest instruction to come up with solution
- Repeat until User is satisfied

Mechanism 2: Memory Stream (Math)

- Stores past messages of user and assistant



- User's message I , Assistant's message A
- Message M , Decision function Ψ , Decision by Assistant S
 - $M_t = \langle (I_1, A_1), (I_2, A_2), \dots, (I_t, A_t) \rangle$
 - $S_t \leftarrow \Psi(I_t, A_t)$
- Use past memory to condition future output
 - $I_{t+1} = A(M_t, S_t)$
 - $A_{t+1} = I(M_t, I_{t+1}, S_t)$
- Add this future output to the corresponding memories
 - $M_{t+1} = M_t \cup (I_{t+1}, A_{t+1})$
 - $S_{t+1} = S_t \cup \Psi(I_{t+1}, A_{t+1})$

Mechanism 3: Self-reflection

- Talking over with another agent can help identify errors

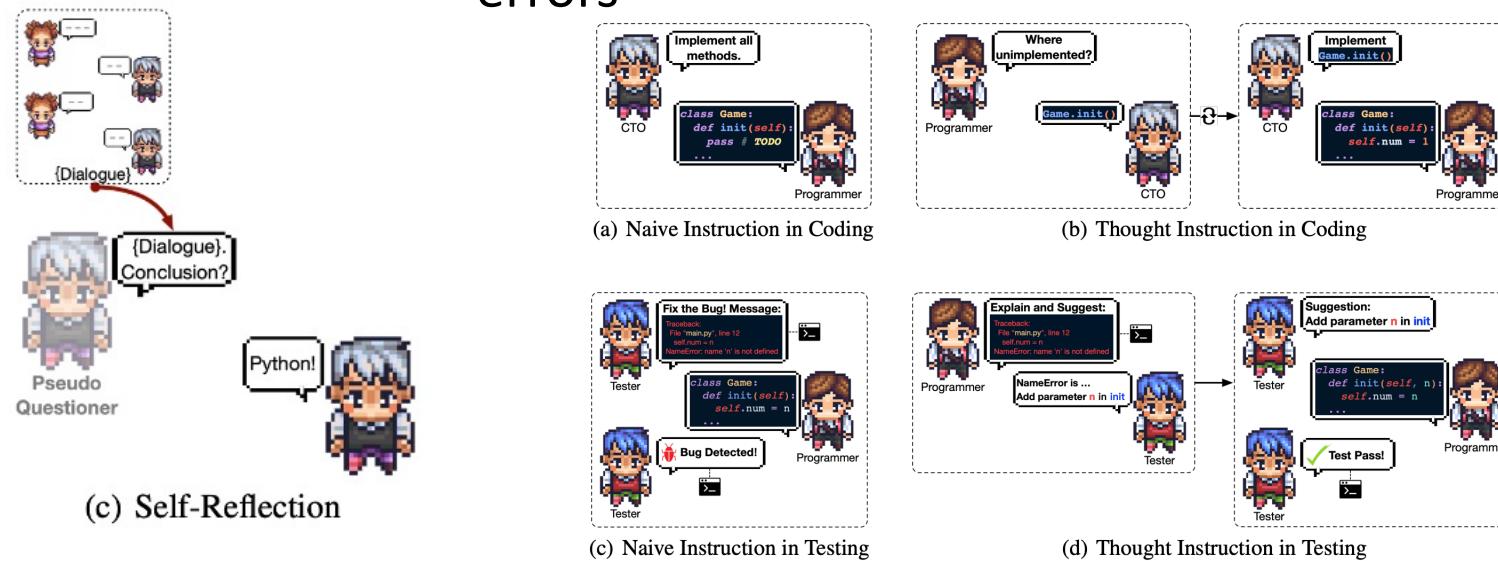


Figure 4: The thought instruction mitigates code hallucinations during the coding and testing phases. Instead of providing generic instructions, thought instruction involves role swapping to inquire about unimplemented methods or explain feedback messages caused by bugs. This step allows for a clearer understanding of the existing code and identifies the specific gaps that need to be addressed. By gaining this awareness, the roles can then switch back, and the instructor can provide more specific instructions to guide the programmer accurately.

Results

Results

- The experiment analyzed all the software produced by CHATDEV in response to 70 user requirements.
- On average, CHATDEV generated **17.04 files** per software, alleviated potential code vulnerabilities caused by code hallucinations **13.23 times**, had a software production time of **409.84 seconds**, and incurred a manufacturing cost of **\$0.2967**.

Error Correction (Reflection) via Conversation

- Discussions between a reviewer and a programmer led to the identification and modification of nearly **twenty types of code vulnerabilities**
- Discussion between a tester and a programmer resulted in the identification and resolution of more than **ten types of potential bugs**

Overall Generation Statistics

Table 1: The statistical analysis of CHATDEV’s software development, including minimum (Min), maximum (Max), and average (Avg.) values for various aspects.

	Min	Max	Avg.
# Code Files	2.00	8.00	4.26
# Asset Files	0.00	21.00	8.74
# Document Files	4.00	5.00	4.04
# Lines of Source Codes	39.00	359.00	131.61
# Lines of Dependencies	1.00	5.00	2.90
# Lines of User Manual	31.00	232.00	53.96
# Version Updates	5.00	42.00	13.23
# Software Re-development	1.00	5.00	1.40

Pros and Cons

Is the whole system of differentiated agents really that important?

Why it is good

- Structure of generation is from broad to specific, helps to guide generation
- Two agents iterating is like doing self-reflection and helps to identify errors

Why it may be bad

- Can code be generated that is very different from the training distribution?
- Currently, all modular codes are part of the same prompt. What if the program involves many parts and cannot fit in the context?"
- Multiple modular codes generated may not even refer to each other. They do not have an overarching framework to follow!
- Indeed: Only **86.66%** of software were executed flawlessly
 - Half is external dependency issues (incorrect versioning, unavailable in the cloud)
 - Half is token length limit issue

Why it may be bad (Part 2)

- Did not really see a sandbox testing step executed to fix errors
 - Code Reviewing is done just by "observing" the code
- Specialisation of agents is just done through prompt. Each agent is not given their own tools/APIs to use to customize to their jobs. Each agent also does not have memory to remember what they did earlier to avoid making the same mistakes.

Some Artifacts

- “We noticed occasional instances of repetitive expressions of gratitude in the dialogue, even after reaching a consensus and making decisions”
 - My opinion: Likely due to RLHF
- “... agents’ limited ability to autonomously determine specific implementation details, often resulting in multiple rounds of lengthy discussions”
 - Chat-based format may not be that ideal for all tasks

Is this really necessary?

Designing: Programming Language

CEO (Chief Executive Officer) : CTO (Chief Technology Officer)

- : To satisfy the new user's demand and make a desktop application realizable, you should propose a concrete programming language.
- : Python with the Pygame library is a suitable choice for implementing a basic Gomoku game.
- : Great! Python is a versatile and widely-used programming language that will be suitable for implementing a basic Gomoku game. Its simplicity and readability make it a popular choice for beginners and experienced developers alike.
- : Let's get started with the development of the basic Gomoku game in Python using the Pygame library!
- : <END> Python.

ChatDev

⚡ Default (GPT-3.5)

C

I want to create a Gomoku game.

Output the following json without explanation

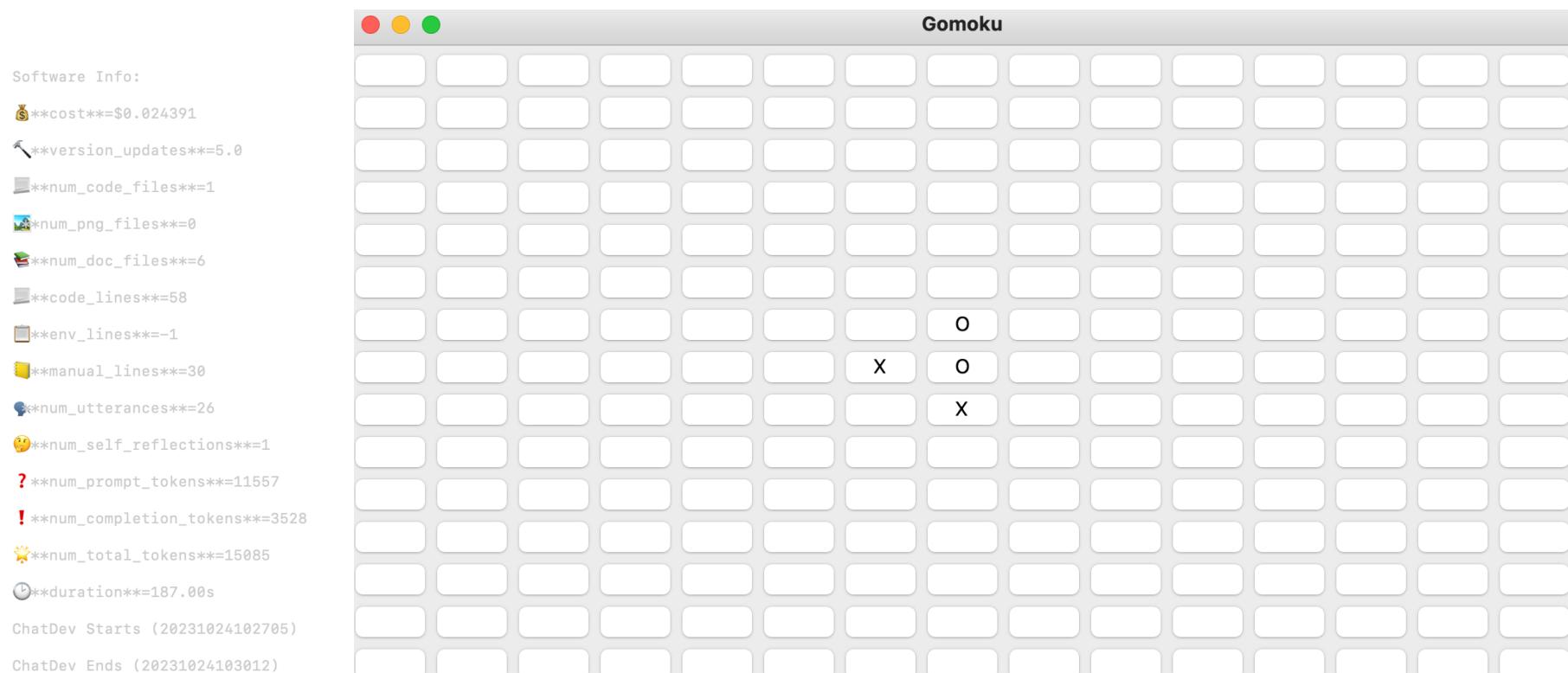
```
{'programming language': 'programming language to create it',  
 'modules': 'corresponding modules to import'}
```



```
{  
  "programming language": "Python",  
  "modules": "pygame"  
}
```

Mine

My own Gomoku game!



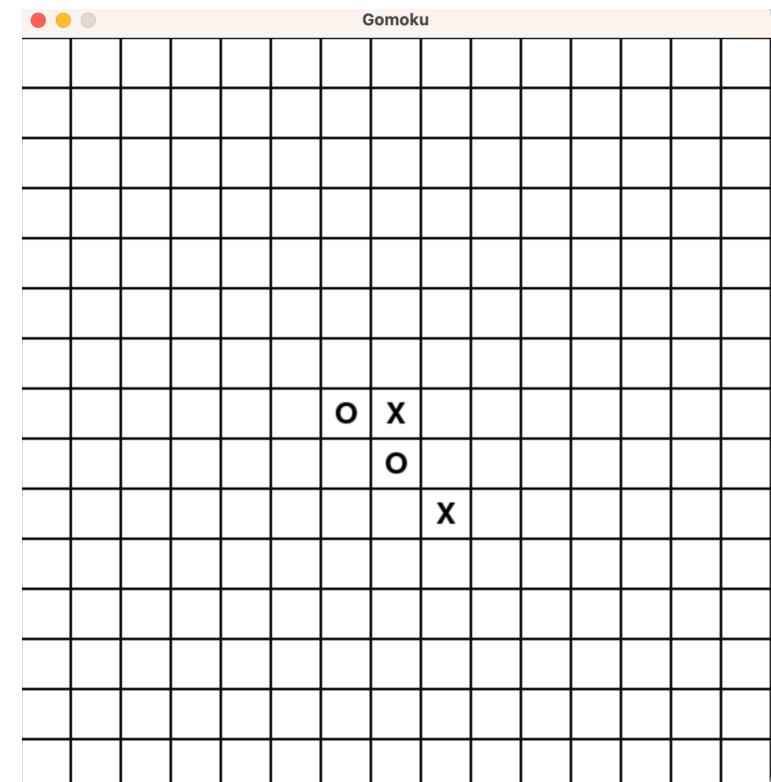
How to create my own Gomoku Game using ChatGPT

⚡ Default (GPT-3.5)

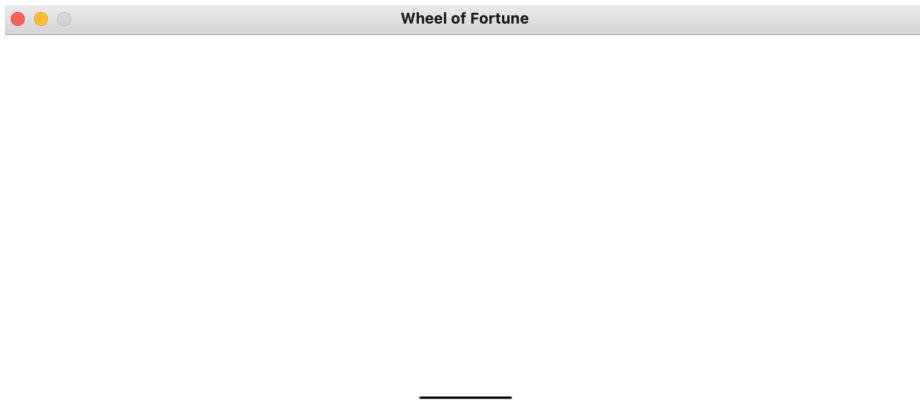
C

Create me a Gomoku game in pygame

Cost: < \$0.01 cents



My own Wheel of Fortune Game! (Nope, not a bug – it compiles, but just no game)



Software Info:

- cost=\$0.048696
- version_updates=9.0
- num_code_files=4
- num_png_files=0
- num_doc_files=7
- code_lines=169
- env_lines=1
- manual_lines=41
- num_utterances=32
- num_self_reflections=1
- num_prompt_tokens=20761
- num_completion_tokens=8777
- num_total_tokens=29538
- duration=468.00s

ChatDev Starts (20231024104450)

ChatDev Ends (20231024105238)

```
# Initialize pygame and create the game window
pygame.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Wheel of Fortune")
clock = pygame.time.Clock()

# Game class
class Game:
    def __init__(self):
        self.running = True
        self.current_word = ""
        self.guessed_letters = set()
        self.load_word()
        self.player = Player("Player 1")
        self.wheel = Wheel()
    def load_word(self):
        # Load a random word from a word list
        word_list = ["apple", "banana", "cherry", "orange"]
        self.current_word = random.choice(word_list)
    def handle_events(self):
        # Handle game events
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                self.running = False
            elif event.type == pygame.KEYDOWN:
                if event.key >= pygame.K_a and event.key <= pygame.K_z:
                    letter = chr(event.key)
                    self.player.guess_letter(letter)
    def update(self):
        # Update game state
        pass
    def draw(self):
        # Draw game elements
        screen.fill(WHITE)
        word_display = self.player.display_word(self.current_word)
        font = pygame.font.Font(None, 36)
        text = font.render(word_display, True, BLACK)
        text_rect = text.get_rect(center=(WIDTH // 2, HEIGHT // 2))
        screen.blit(text, text_rect)
        pygame.display.flip()
    def run(self):
        # Game loop
        while self.running:
            self.handle_events()
            self.update()
            self.draw()
            clock.tick(FPS)
# Create and run the game
game = Game()
game.run()
# Quit pygame
pygame.quit()
```

How to create my own Wheel of Fortune Game using ChatGPT

C

Create me Wheel of Fortune using PyGame

⚡ Default (GPT-3.5)

Cost: < \$0.01 cents

```
UnboundLocalError
Cell In[1], line 71
  68     pygame.quit()
  70 if __name__ == "__main__":
----> 71     main()

Cell In[1], line 61, in main()
  58 screen.fill(BACKGROUND_COLOR)
  59 draw_wheel()
----> 61 if spinning:
  62     pygame.time.delay(1000) # Simulate spinning
  63     spinning = False

UnboundLocalError: cannot access local variable 'spinning' where it is not associated with a value
```

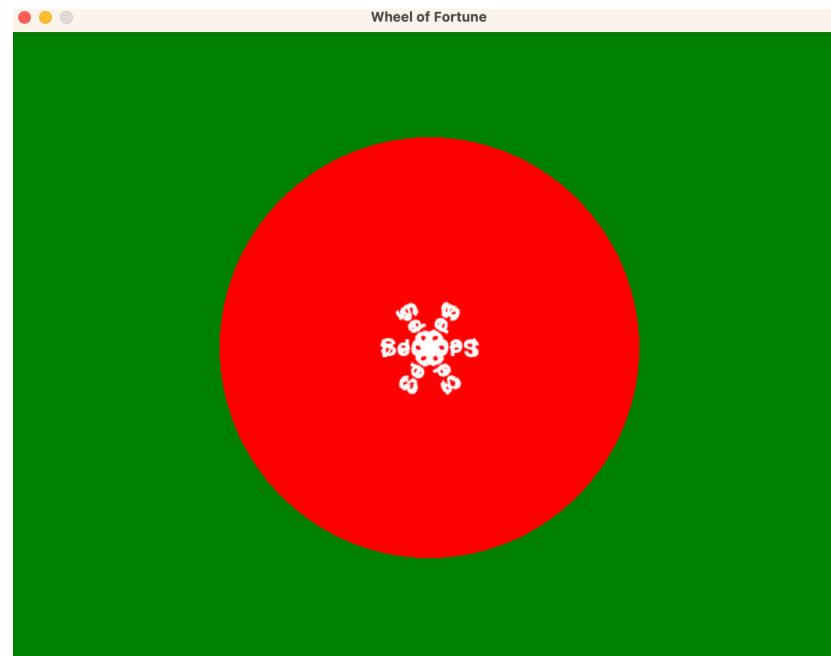
Oops, error message. Let's prompt the error message too!

How to create my own Wheel of Fortune Game using ChatGPT

C

```
-----  
UnboundLocalError          Traceback (most recent call last)  
Cell In[1], line 71  
    68     pygame.quit()  
    70 if __name__ == "__main__":  
--> 71     main()  
  
Cell In[1], line 61, in main()  
    58 screen.fill(BACKGROUND_COLOR)  
    59 draw_wheel()  
--> 61 if spinning:  
    62     pygame.time.delay(1000) # Simulate spinning  
    63     spinning = False  
  
UnboundLocalError: cannot access local variable 'spinning' where it is not associated with a  
value  
  
Generate me the updated full Python code block.
```

Cost: < \$0.01 cents



How I think it can be done (Broad steps)

C

I would like to create a minimalist Wheel of Fortune Game with PyGame. List me the files I would need to create and their functions.



Creating a minimalist Wheel of Fortune game with PyGame involves several files and functions to manage different aspects of the game. Below is a list of files and their associated functions:

1. `main.py`:

- This is the main script where you'll initialize the game, create the game loop, and handle user input.
- Functions:
 - `main()`: The main game loop.
 - `initialize_game()`: Initialize the game, load assets, and set up game variables.
 - `handle_events()`: Handle user input events (e.g., spinning the wheel, guessing letters).
 - `update()`: Update the game state (e.g., check guesses, update scores).
 - `render()`: Draw the game on the screen.

2. `wheel.py`:

- This file defines the spinning wheel of fortune.
- Functions:
 - `Wheel()`: Class representing the spinning wheel.
 - `spin()`: Spin the wheel and determine the prize.

3. `puzzle.py`:

- This file contains the logic for the game's word puzzles.
- Functions:
 - `Puzzle()`: Class representing a word puzzle.
 - `set_puzzle()`: Set a new puzzle for the game.
 - `guess_letter()`: Check if a guessed letter is correct and update the puzzle.
 - `is_solved()`: Check if the puzzle has been solved.

4. `player.py`:

- This file defines the player class.
- Functions:
 - `Player()`: Class representing a player.
 - `add_score()`: Update the player's score.
 - `get_score()`: Retrieve the player's score.

5. `assets.py`:

- Store all game assets (e.g., images, sounds) in this file for easy loading.
- Functions:
 - `load_assets()`: Load game assets into memory.

6. `config.py`:

- Store configuration settings for the game, such as screen dimensions, colors, and fonts.

7. `words.py`:

- Contains a list of word puzzles that can be used in the game. These puzzles should be stored in a list or a file that can be imported.

How I think it can be done (Specific steps)

The image shows a terminal window titled "Wheel of Fortune". Inside the window, the text "PYTHON I_ GREAT" is displayed in white, followed by "You Win!" in white. At the bottom, the text "Guessed Letters: G, D, Y, F, E, C, I, R, T, O, H, P, N, A" is shown in white. To the left of the terminal window, there is a blue button with a white "C" icon and the text "Create me main.py". Below this button, a yellow box contains the text "Cost: < \$0.01 cents".

Questions to Ponder

- Will a group of agents do better than just having a single one do the task?
- Can we replace the entire workflow with just one agent prompted sequentially with multiple contexts?
- How can we encode memory for each agent separately to increase their expertise in the area so they can grow in experience and not repeat earlier mistakes?