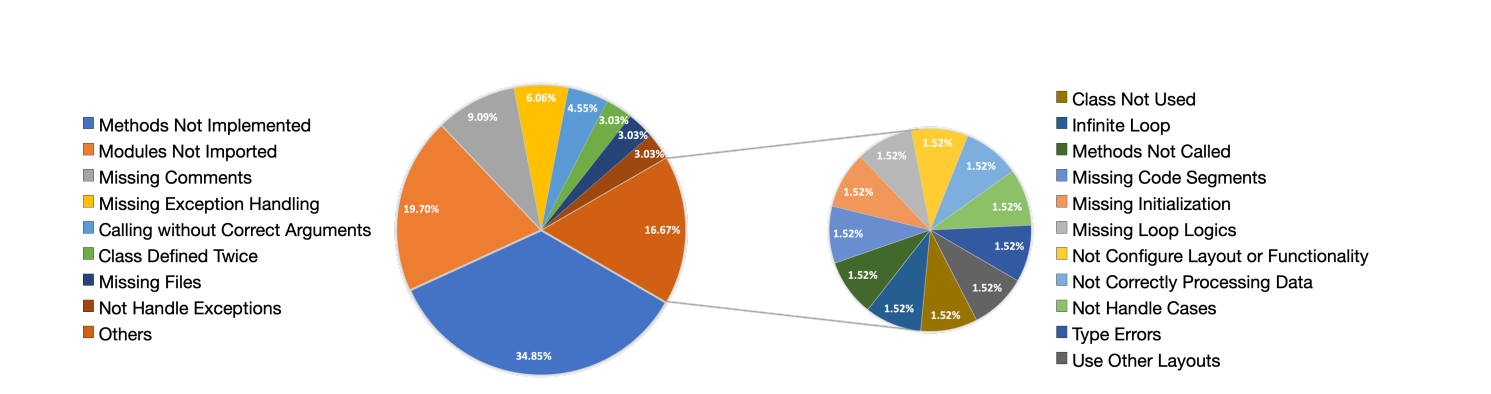
# Chatdev analysis

We effortlessly set up the sandbox environment in our experiments by directly installing the required software dependencies. Subsequently, we executed the generated software using the main function. Remarkably, approximately 86.66% of the software systems executed flawlessly, showcasing the robustness and reliability of our developed software. However, a small fraction, 13.33% of the software, encountered execution failures.

We conducted a duration analysis to examine the software production time for different request prompts using ChatDev. The variability in development times across prompts reflects the varying complexity and clarity of the assigned tasks. The graph in Figure 6 provides a visual representation of this distribution. The longest software production duration, represented by the tallest bar on the left side of the graph, was 1030.00 seconds. This extended time was due to extensive dialogue and communication between the reviewer and programmer, leading to a detailed modification scheme.

We noticed occasional instances of repetitive expressions of gratitude in the dialogue of the chat chain, even after reaching a consensus and making decisions.

We monitored API interactions and token usage during software production in ChatDev. On average, ChatDev requires 36,902.23 prompt tokens, 11,567.37 completion tokens, and a total of 48,469.60 tokens to develop a single software. The average total cost in software production is approximately $0.15693 . To determine the overall cost of software development with ChatDev, we also consider the cost of designer-produced images. The average designer cost is $0.1398 per software for each software production involving 8.74 graphics creations on average. Thus, the average software development cost at ChatDev is $0.2967, significantly lower than traditional custom software development companies’ expenses.



Even though ChatDev offers a novel paradigm for software development that is training-free, efficient, and cost-effective, we recognize the presence of potential risks and limitations that require further investigation and resolution. Even when we set the temperature parameter of the large language model to a very low value, we observe inherent randomness in the generated output. Consequently, each software produced may vary between different runs. As a result, this technology is best suited for open and creative software production scenarios where variations are acceptable. Moreover, there are instances where the software fails to meet the users’ needs. This can be attributed to unclear user requirements and the inherent randomness in text or code generation. While the designer agent is capable of creating images [35], it is important to acknowledge that the directly generated images may not always enhance the GUI’s aesthetics. At times, they may introduce excessive complexity, which can hinder user experience. This is primarily because each image is generated independently, lacking direct visual correlation. To address this, we have provided the option for users to customize the GUI as a system hyperparameter, allowing them to decide whether to enable this feature or not. Additionally, the large language model may exhibit inherent biases [30], leading to the generation of code patterns that do not necessarily align with the problem-solving thinking of real programmers. Regarding risks, it is important to note that existing large language models are not fully tuned to be harmless, making them vulnerable to potential misuse by malicious users for harmful purposes. Furthermore, the generated software currently lacks malicious intent identification for sensitive file operations. Therefore, users are advised to conduct their own code review before running the software to prevent any unnecessary data loss. Additionally, the assessment of our ChatDev framework’s software-level task completion capabilities presents formidable challenges, owing to the vast scope and heterogeneous nature of the generated tasks. This mandates the active participation of a multitude of domain experts. Although the study may potentially help junior programmers or engineers in real world, it is challenging for the system to generate perfect source code for high-level or large-scale software requirements. This difficulty arises from the agents’ limited ability to autonomously determine specific implementation details, often resulting in multiple rounds of lengthy discussions. Additionally, large-scale software development proves challenging for both reviewers and testers, as it becomes difficult to identify defects or vulnerabilities within the given time constraints.