AGILECODER: Dynamic Collaborative Agents for Software Development based on Agile Methodology

Anonymous ACL submission

Abstract

Software agents are emerging as powerful tools for tackling complex software engineering tasks. However, existing approaches often oversimplify development workflows, assuming basic models that lack the nuances of real-world software processes. Moreover, they frequently include entire codebases in their instructions, leading to inefficiencies when working with large-scale projects. To overcome these challenges, we introduce **AGILECODER**, a multi-agent system that incorporates Agile Methodology (AM) principles, assigning specialized agents to roles such as Product Manager, Developer, and Tester for collaborative, iterative development. AGILECODER structures work into sprints, enabling *incremental* progress based on user input. A standout feature, the Dynamic Code Graph Generator, continuously builds a Code Dependency Graph as the codebase evolves, allowing agents to gain a deeper understanding of the structure for more precise code generation and efficient modifications. We evaluate AGILECODER on two fronts: (1) code generation benchmarks, including HumanEval and MBPP, and (2) real-world software development scenarios. The results show that AGILECODER outperforms existing systems like ChatDev and MetaGPT, setting a new standard for multi-agent systems in software engineering. The source code is available at `https://anonymous.4open.science/r/AgileCoder-Submission`.

1 Introduction

Autonomous software agents leveraging Large Language Models (LLMs) offer significant opportunities to enhance and replicate software development workflows (Qian et al., 2023; Hong et al., 2024; Tang et al., 2024; Zhou et al., 2023; Huang et al., 2023). These agents simulate human software development processes, including design, implementation, testing, and maintenance. MetaGPT (Hong et al., 2024) encodes Standardized Operating Procedures (**SOPs**) into software development tasks, while ChatDev (Qian et al., 2023) creates a virtual chat-powered technology company; both follow the classic **waterfall** model. However, these approaches oversimplify the workflow, failing to reflect the dynamic and iterative nature of real-world software development, where approximately 70% of professional teams adopt **Agile Methodology (AM)** (Agi, 2024). Moreover, these methods overly depend on LLMs for decision-making and managing code generation, proving inadequate for handling the complexity of entire software repositories, especially when considering repository-level code understanding and generation. They often rely on straightforward prompting strategies that involve including the entire source code as input to the model. While this strategy may work in smaller projects, it quickly becomes inefficient and ineffective in larger, more complex codebases. LLMs can become overwhelmed by irrelevant context, leading to slower processing times and less accurate results. To address these limitations, we propose AGILECODER, a novel multi-agent software development framework based on Agile Methodology (AM). AGILECODER mimics the AM workflow and adapts it to a multi-agent framework context, allowing for dynamic adaptability and iterative enhancement capabilities. We also introduce a static-analysis-based module called Dynamic Code Graph Generator, which creates a Code Dependency Graph (CDG) that updates whenever the code changes. The CPG serves as a reliable source for agents to retrieve relevant contexts, enabling precise refinements of the software in each sprint.

To evaluate the efficacy of AGILECODER, we conduct an extensive evaluation on two well-known benchmarks, HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021a) and a novel software development benchmark, ProjectDev. ProjectDev is specifically designed for software development

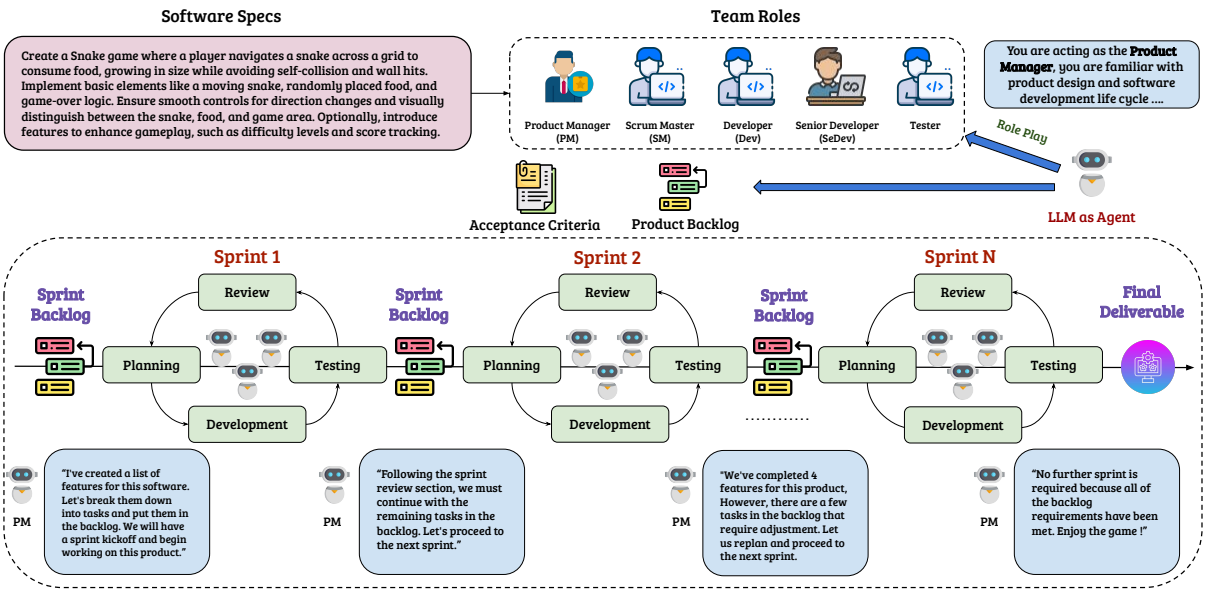

Figure 1: An overview of AGILECODER

tasks to complement existing benchmarks like HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021a), which contain simpler competitive problems and may not fully capture the complexity of multi-agent systems. While MetaGPT (Hong et al., 2024) and ChatDev (Qian et al., 2023) present related benchmarks, SoftwareDev and Software Requirement Description Dataset (SRDD), respectively, neither is publicly available. Therefore, we craft ProjectDev to more effectively assess multi-agent systems, with further details provided in the Appendix A.3. Experimental results demonstrate that AGILECODER achieves the best scores of pass@1 across the first two datasets. For instance, using GPT-3.5 Turbo as the backbone model, AGILECODER improvements of 7.71% and 6.19% compared to the MetaGPT (Hong et al., 2024). Furthermore, AGILECODER exhibits outstanding performance on ProjectDev compared to ChatDev (Qian et al., 2023) and MetaGPT(Hong et al., 2024) in generating executable programs meeting user requirements. In short, our main contributions are summarized as follows:

(1) We introduce AGILECODER, a novel multi-agent software development framework inspired by Agile methodology, which emphasizes effective communication and incremental development among agents. This framework allows for the inheritance of outputs across sprints, enabling continuous refinement and increasing the success likelihood of the final products.

(2) We integrate a static analysis method into the multi-agent workflow through the Dynamic Code Graph Generator (DCGG), dynamically producing a Code Dependency Graph (CDG). This graph captures the relationships among code components as the codebase evolves, providing a reliable source for agents to retrieve relevant contexts and thereby enhancing the quality of the produced software. Our evaluation demonstrates a significant performance increase in real-world benchmarks when utilizing contexts retrieved from the CDG.

(3) Our evaluations confirm that AGILECODER achieves new state-of-the-art (SOTA) performance on established benchmarks like HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021a), as well as our proposed benchmark for real-world software development, named ProjectDev. This framework outperforms recent SOTA models such as MetaGPT (Hong et al., 2024) and ChatDev (Qian et al., 2023), demonstrating its efficacy in practical software development scenarios.

2 Related Work

Deep Learning for Automated Programming In recent years, applying deep learning to automated programming has captured significant interest within the research community (Balog et al., 2016; Bui and Jiang, 2018; Bui et al., 2021; Feng et al., 2020; Wang et al., 2021; Allamanis et al., 2018; Bui et al., 2023; Guo et al., 2020, 2022). Specifically, Code Large Language Models (CodeLLMs) have emerged as a specialized branch of LLMs, fine-tuned for programming tasks (Wang

et al., 2021, 2023; Feng et al., 2020; Allal et al., 2023; Li et al., 2023; Lozhkov et al., 2024; Guo et al., 2024a; Pinnaparaju et al., 2024; Zheng et al., 2024; Roziere et al., 2023; Nijkamp et al., 2022; Luo et al., 2024b; Xu et al., 2022; Bui et al., 2022). These models have become foundational in an industry that offers solutions like Microsoft Copilot, and TabNine, excelling at solving competitive coding problems from benchmarks such as HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021b), and APPs (Hendrycks et al., 2021). Despite achieving good results with benchmark tasks, these models often struggle to generate real-world software that requires complex logic and detailed acceptance criteria, which are essential for practical applications (Hong et al., 2024; Qian et al., 2023).

LLMs-based Multi-Agent Collaboration for Software Development Recently, LLM-based autonomous agents have gained considerable interest from both industry and academia (Wang et al., 2024; Guo et al., 2024b; Du et al., 2023). In software development, agent-centric systems specialized in coding tasks have driven significant advancements (Hong et al., 2024; Qian et al., 2023; Chen et al., 2023; Huang et al., 2023; Zhong et al., 2024; Lin et al., 2024; Yang et al., 2024). These systems assign distinct roles such as Programmer, Reviewer, and Tester, each responsible for a specific phase of the code generation workflow, enhancing both quality and efficiency. Benchmarks have been developed alongside these systems to evaluate their ability to handle real-world software engineering tasks. For example, MetaGPT introduces SoftwareDev (Hong et al., 2024), a suite of diverse software requirements that tasks agents with developing complete software solutions. Similarly, we compile a set of software requirements to evaluate AGILECODER 's capability in real-world software development. SWE-Bench (Jimenez et al., 2024), another prominent benchmark, is designed to evaluate multi-agent systems in handling real-world GitHub issues, focusing on tasks related to software maintenance and evolution. ***However, SWE-Bench is not directly applicable to* AGILECODER *, as our work targets the generation of complete software from user tasks, more aligned with systems like ChatDev and MetaGPT (Qian et al., 2023; Hong et al., 2024), rather than focusing on software maintenance.***

Repository-Level Code Understanding & Generation Generating code at the repository level is a significant challenge for Large Language Models (LLMs) in real-world software engineering tasks (Shrivastava et al., 2023a,b; Bairi et al., 2023; Zhang et al., 2024; Agrawal et al., 2023; Phan et al., 2024). Real-world codebases are complex, with interconnected modules, and as the context size increases, LLMs face limitations. This has led to research on selecting relevant contexts (Luo et al., 2024a; Shrivastava et al., 2023a; Liu et al., 2024) and optimizing their use. Software agents, like ChatDev and MetaGPT, aim to generate fully functional, executable software comprising various files, classes, and modules, rather than just solutions for simple tasks like those in HumanEval (Chen et al., 2021) or MBPP (Austin et al., 2021b). This requires agents to understand all existing contexts, including files, classes, functions, and libraries, when generating code or fixing bugs. However, this need for comprehensive repository-level code understanding and generation has often been overlooked in prior research.

3 AGILECODER: An Agentic Framework for Software Development

Figure 1 presents an overview of AGILECODER, which employs multiple agents in roles such as Product Manager, Scrum Master, Developer, Senior Developer, and Tester, collaborating through an Agile Methodology-inspired workflow. The development process incorporates the Execution Environment for running code during testing and the Dynamic Code Graph Generator (DCGG) (Figure 2) for dynamically generating the Code Dependency Graph whenever the code is updated. The Execution Environment provides tracebacks to agents for code refinement, while the DCGG enables agents to retrieve relevant contexts for accurate code generation and correction. Detailed descriptions of these modules are provided in the following Sections.

3.1 Agent Roles

The roles of each agent are defined as follows:

- Product Manager (PM): Takes requirements from users and creates a product backlog, which includes development tasks and acceptance criteria.
- Scrum Master (SM): Provides backlog feedback to the PM to enhance sprint achievability.
- Developer (Dev): Primarily focuses on developing tasks, which include generating and refactoring code based on feedback from other agents.

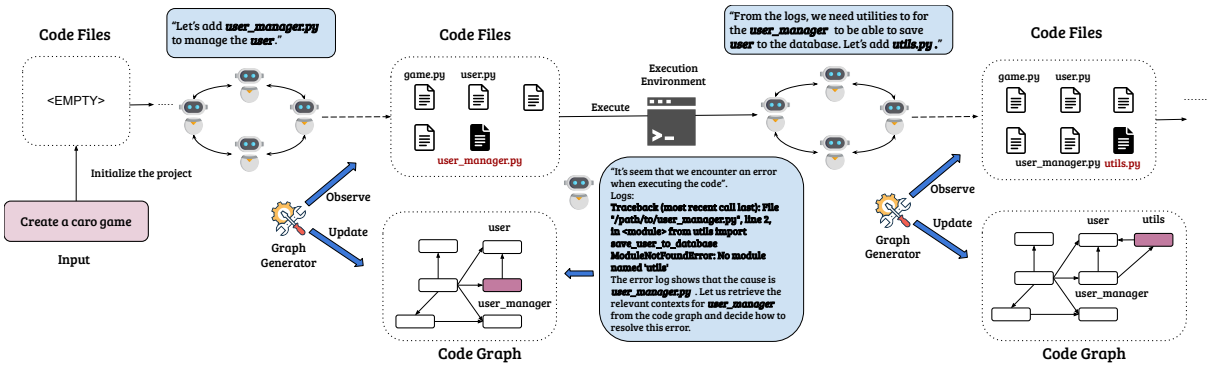

Figure 2: Illustration of how Dynamic Code Graph Generator (DCGG) contributes to AGILECODER during the generation of a Python application.

- Senior Developer (SD): Reviews code generated by developers, provides feedback for code refactoring, and ensures quality control.
- Tester: Generates test cases and scripts to validate the code developed by the developers.

3.2 Overview of Workflow

The workflow starts with the Product Manager (PM) initiating backlog planning after receiving user requirements. The PM writes tasks and acceptance criteria in the backlog. The Scrum Master (SM) reviews the backlog, assesses task feasibility, and may request revisions from the PM. The output is the *product backlog*, capturing all tasks required for final deliverables. Once planning is complete, the SM initiates development with a sprint. Each sprint includes Planning, Development, Testing, and Review phases. During Planning, the PM selects tasks from the Product Backlog for the Sprint Backlog. The remaining phases involve sub-phases where agent pairs collaborate on tasks (details in subsequent sections). After all phases, the PM evaluates progress to decide if the software is ready for delivery. If not, the next sprint begins with Planning. This repeats until the PM decides the software is deliverable, at which point a termination signal concludes the development pipeline.

3.2.1 Planning Phase

At the start of each sprint, the Product Manager (PM) drafts a plan including tasks selected from the product backlog, defined acceptance criteria, and insights from reviews of previous sprints, if applicable. The Scrum Master provides feedback to refine this plan, ensuring that it aligns with the project's current objectives and constraints. The output of this phase is the **sprint backlog**, outlining the scope of tasks to be completed within the sprint.

3.2.2 Development Phase

Following the planning phase, the PM directs the Developer to begin implementation. To enhance clarity for other agents, the Developer is required to annotate each method/function with docstrings. However, due to potential inaccuracies from LLM hallucinations (Manakul et al., 2023), the code produced might not always align with the sprint backlog and acceptance criteria. To mitigate this, the Senior Developer employs a static review process inspired by peer review practices. This review focuses on identifying bugs, logic errors, and edge cases, subsequently providing feedback for corrections. The review process is structured into three sequential steps to manage complexity and improve the effectiveness of feedback. The first step involves checking for basic errors like empty methods and missing import statements, followed by ensuring that source code fulfills the sprint backlog with the final step confirming that source code meets acceptance criteria and is free of bugs.

By breaking down the code review into these structured steps, it becomes feasible for LLMs to conduct thorough static analyses and provide actionable feedback, thereby enhancing the accuracy and reliability of the development phase. An ablation study can be found in the Appendix A.2.1.

3.2.3 Testing Phase

Despite thorough review, error-free code cannot be guaranteed due to LLM hallucinations. Therefore, a tester is employed to write test suites and implement a testing plan, furnishing real-time feedback to the Developer for iterative code refinement.

Writing Test Suites During a sprint, we inherit source code from previous sprints and implement new features. While existing code undergoes thor-

ough review and testing, new code lacks such scrutiny, necessitating the creation of test cases to ensure its functional correctness. We utilize the code graph G created by the DCGG (Section 3.3) and the list of changed files $\mathcal{F}$ to find files requiring testing. This process can be formally described as $\bigcup f(n_i) \cup \{n_i\}, n_i \in \mathcal{F}$, where f returns ancestor nodes in the graph G of an input node. For instance, in Figure 2, if the file *user_manager.py* undergoes any changes, we should recheck its functional correctness along with that of its ancestor files, but not *user.py*. The Tester is then responsible for writing test suites for all necessary files.

Writing A Testing Plan After writing test cases, we have multiple testing scripts that must be executed in a specific order to avoid inconsistency among code files and unnecessary costs. For instance, in Figure 2, the file *user_manager.py* depends on the file *user.py*, making it illogical to test *user_manager.py* before *user.py*. Fortunately, we can obtain a logical testing plan by reversing a ***topological order*** among testing scripts of the code graph G. Furthermore, we want the final software to be executable, so the Tester is required to write commands to evaluate its executability.

Fixing Bugs Once a well-defined testing plan is established, files are iteratively executed according to the plan until issues such as bugs or failed test cases arise. The Developer addresses these issues based on test reports provided by the Tester, repeating the process until the testing plan is completed.

3.2.4 Review Phase

At the end of a sprint, the Tester runs software to write a testing report. The Product Manager then collects the sprint backlog, source code, and the testing report to assess completed, failed, and incomplete tasks. This information accumulates over sprints to form an overall report. The Product Manager then compares this overall report with the product backlog and acceptance criteria to decide whether to conclude the task or plan the next sprint. If concluding, the Scrum Master writes detailed documentation, including how to run and install necessary libraries. If planning another sprint, the Product Manager reviews the product backlog, acceptance criteria, and the current overall report to create the next sprint plan.

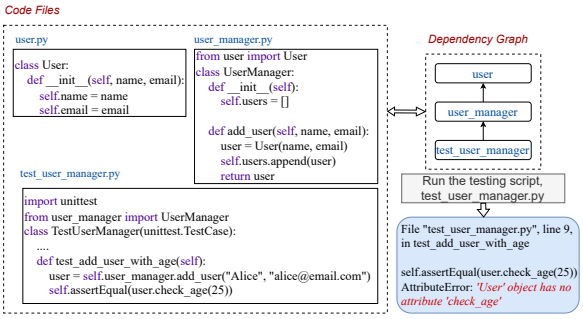

Figure 3: Utilization of DDCG during bug-fixing to enable more efficient modifications and accurate code generation.

3.3 Dynamic Code Graph Generator for Context-Aware Code Retrieval

The Dynamic Code Graph Generator (DCGG) is a critical module in our system that dynamically creates and updates a code dependency graph, denoted as G. This graph effectively models the relationships among code files, facilitating efficient context-aware code retrieval. In G, each node represents a code file, and each edge indicates a dependency relation, such as the relationship between *user_manager.py* and *user.py* shown in Figure 2. We primarily capture *import* relationships to maintain the graph's simplicity and efficiency. As the codebase evolves, whether through the addition of new features or corrections of bugs, G is updated to reflect the current state accurately. This is essential for maintaining the integrity of the codebase and ensuring that all changes are properly documented and integrated. Additionally, files modified during a sprint, identified as $\mathcal{F}$, are recorded and linked within G to track all affected dependencies. In summary, the DCGG serves two primary functions:

1. **Assisting in Writing Test Cases and Testing Plan** When new code has been generated, DDCG ensures that only influenced files need testing. Also, a reasonable testing plan is always obtained by using the G.

2. **Providing Context for Code Repair:** When an error occurs during code execution, the agent traces back to identify the relevant files using traceback analysis. It retrieves cross-file contexts from the code graph to pinpoint the source of the error, ensuring effective code retrieval.

Existing approaches, such as those employed by ChatDev (Qian et al., 2023) and MetaGPT (Hong et al., 2024), often involve loading the entire code-

base into the LLMs, which becomes impractical as the codebase grows beyond the token limits of LLMs. DCGG addresses this limitation by maintaining a dependency graph that allows the agent to selectively retrieve relevant information, thereby optimizing the LLM's token usage and maintaining the relevance and accuracy of the information provided to the developers or agents. For instance, in Figure 3, AGILECODER analyzes the traceback and utilizes DCGG to trace the issue to the *user* node which corresponds to the *user.py*, enabling the most relevant files included in the instructions for more efficient and precise code generation, whereas other methods often include the entire codebase, leading to potential inefficiencies.

This design approach not only improves the efficiency of code generation and debugging but also ensures that the development process is streamlined and focused, avoiding the overload of irrelevant data and enhancing the overall accuracy of the development workflow. Figure 2 illustrates how DCGG updates in response to codebase changes, such as the addition of new classes or relationships, thereby maintaining an up-to-date and accurate reflection of the code structure.

3.4 Communication Mechanism Among Agents

Our system employs a dual-agent conversation design across all phases, simplifying the interaction model to just two roles: an instructor and an assistant. This structure avoids the complexities associated with multi-agent topologies and streamlines the consensus process. The instructor is responsible for providing clear instructions and guiding the flow of the conversation toward the completion of specific tasks, while the assistant uses their skills and knowledge to execute the tasks, with interactions continuing until a consensus is reached.

To ensure continuity and coherence in the dialogue between messages, we employ a Message Stream $\mathbf{S}$. This stream acts as a working memory that stores all exchanged messages, allowing agents to create responses that seamlessly connect with the conversation history. Formally, if i_t and a_t are messages produced by the instructor and assistant at time step t respectively, then $\mathbf{S}_t$ is defined as $\mathbf{S}_t = [(i_1, a_1), (i_2, a_2), ..., (i_t, a_t)]$.

At the next time step $t+1$, the instructor reviews $\mathbf{S}_t$ to assess alignment of action a_t with provided instructions before issuing further guidance i_{t+1}. Simultaneously, the assistant, upon receiving the new instruction i_{t+1}, crafts a suitable response a_{t+1}. If a_{t+1} satisfies the termination criteria or if the interaction reaches a predefined limit of exchanges, the dialogue concludes. The final message a_{t+1} is stored for future reference, ensuring retention of pertinent and constructive information.

Communication Protocol In line with established practices from prior research (Qian et al., 2023), our communication interface utilizes unconstrained natural language. This flexibility allows for easy modifications of prompts, such as adjusting output constraints or changing formats, without significant system-wide impacts. Moreover, AGILECODER incorporates a technique known as prompt engineering at the onset of each conversation to optimize understanding and task execution. This initial setup ensures that both agents are well-informed about the task requirements and objectives, facilitating the generation of relevant responses aimed at successfully completing the tasks. The agents proceed autonomously without human intervention until consensus is achieved.

Global Message Pool To facilitate smooth information flow throughout the system, a global message pool within a shared environment stores the outputs from all conversations. This environment also logs the status of tasks—whether they are completed, pending, or failed—providing a rich context for the agents to make informed decisions. However, given the potential volume of information, direct exposure of all data to the agents could lead to information overload. Therefore, each conversation accesses only the relevant segments of data from the global message pool necessary for task resolution. For instance, while the Product Manager may focus on the product backlog and task statuses for sprint planning, the Developer might need access primarily to specific sections of the code relevant to bug fixes. This targeted access strategy prevents data overload and ensures that all agents operate with the most current and relevant information available.

4 Experiments

4.1 Empirical Results

Datasets For evaluation, we selected two types of datasets. The first includes well-known benchmarks for assessing code generation capabilities in CodeLLMs: HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021a). More experi-

Category		Model	Dataset Performance	
			HumanEval	**MBPP**
LLMs (prompting)		CodeGeeX-13B	18.9	26.9
		PaLM Coder-540B	43.9	32.3
		DeepSeeker-33B-Inst	79.3	70.0
		GPT-3.5 Turbo	60.3	52.2
		Claude 3 Haiku	75.9	80.4
		GPT 4	80.1	80.1
LLMs-based Agents	with GPT-3.5 Turbo	ChatDev	61.79	74.80
		MetaGPT	62.80	74.73
		AgileCoder	**70.53**	**80.92**
	with Claude 3 Haiku	ChatDev	76.83	70.96
		AgileCoder	**79.27**	**84.31**
	with GPT 4	MetaGPT	85.9	87.7
		AgileCoder	**90.85**	-

Table 1: Comparative results on HumanEval and MBPP datasets for various LLMs and LLM-based agents, highlighting performance enhancements achieved through the application of AgileCoder.

mental settings can be found in the Appendix A.1. These benchmarks primarily feature competitive-level problems that do not fully represent the complexities of real-world software development.

To address this gap, we have collected a collection of 30 representative examples of more intricate software development tasks, referred to as ProjectDev. These tasks cover diverse areas such as mini-games, image processing algorithms, data visualization, desktop applications, and web applications. Each task comes with a detailed prompt and requires the system to generate a comprehensive codebase consisting of multiple executable files. We run each task three times and report average numbers. Details about ProjectDev, including the dataset construction process, evaluation steps, all tasks, can be found in the Appendix A.3.

Metrics For HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021a), we adopt the unbiased pass@k metric (Chen et al., 2021), following the approach of prior studies (Hong et al., 2024; Qian et al., 2023), to evaluate the functional accuracy of top-1 generated code. For the ProjectDev dataset, we focus on practical application and evaluate performance through human assessment and statistical analysis. Human evaluation involves checking the executability of the generated software against expected requirements to determine the success rate in meeting those requirements (e.g. if a generated program is executable and meets 4 out of 10 requirements, its executability rate is 40%) and computing the total number of errors (#Errors) when generated programs fail to run. Statistical analysis includes metrics such as runtime, token usage, expenses for all methods, and the average number of sprints (#Sprints) for only AgileCoder.

Baselines We employ SOTA CodeLLMs as baselines, including CodeGen (Nijkamp et al., 2022), CodeGeeX (Zheng et al., 2023), PaLM Coder (Chowdhery et al., 2023), DeepSeek-Coder (Guo et al., 2024a), GPT-3.5, GPT-4 (Achiam et al., 2023), and Claude 3 Haiku (Anthropic, 2024). Given that AgileCoder is a multi-agent system, we also compare it against leading multi-agent systems used for software development tasks, such as MetaGPT (Hong et al., 2024) and ChatDev (Qian et al., 2023).

Statistical Index	ChatDev	MetaGPT	AgileCoder
Executability	38.01	23.57	**59.03**
Entire Running Time (s)	124	**47.8**	402
Avg. Time/Sprint (s)	-	-	187
#Sprints	-	-	2.93
Token Usage	5504	**2775**	29158
Expenses (USD)	0.08	**0.02**	0.37
#Errors	22	50	**0**

Table 2: Results on ProjectDev

Results Table 1 shows that AgileCoder significantly outperforms recent SOTA multi-agent frameworks and CodeLLMs on the HumanEval and MBPP benchmarks. AgileCoder achieves an average improvement of 5.58 and 6.33 in pass@1 over ChatDev and MetaGPT on HumanEval, respectively, with similar improvements on MBPP. Results on ProjectDev (Table 2) further demonstrate AgileCoder's superiority in software de-

velopment tasks. AGILECODER shows substantial improvements in executability over ChatDev and MetaGPT, without producing any non-executable programs. These advantages can be attributed to AGILECODER's incorporation of the planning phase, generated test cases, and efficient code retrieval, which ChatDev and MetaGPT lack. Although AGILECODER requires more tokens and running time due to the inherent complexity of the Agile Scrum methodology, it efficiently completes user tasks in an average of 2.93 sprints. Our experiments consistently demonstrate AGILECODER's superiority across various benchmarks.

4.2 Analysis

Impact of The Number of Sprints We conduct an ablation study to assess the impact of incremental development. Incremental development involves multiple sprints, whereas its removal condenses the process into a single sprint. Results in Table 3 show that incremental development leads to better performance on HumanEval and MBPP across two models, including GPT-3.5 Turbo and Claude 3 Haiku. This advantage stems from inheriting outputs from previous sprints for further refinement and addressing existing issues in subsequent iterations, thereby increasing the likelihood of successful outcomes.

	Model	Dataset	
		HumanEval	MBPP
GPT-3.5 Turbo	AGILECODER	70.53	80.92
	w/o incremental dev.	69.51 (-1.02)	78.45 (-2.47)
	w/o writing test suite	62.20 (-8.33)	75.64 (-5.28)
	w/o code review	68.90 (-1.63)	75.41 (-5.51)
Claude 3 Haiku	AGILECODER	79.27	84.31
	w/o incremental dev.	76.83 (-2.44)	82.20 (-2.11)
	w/o writing test suite	73.17 (-6.10)	79.86 (-4.45)
	w/o code review	75.00 (-4.27)	80.56 (-3.75)

Table 3: Ablation study on the incremental development, code review and writing test suite

Impact of Code Review and Writing Testing Suite The results in Table 3 demonstrate the importance of code review and writing test cases in AGILECODER. Removing either of these tasks from sprints leads to performance degradation. In particular, the absence of generated test cases significantly impacts performance, confirming their role in detecting potential bugs and improving code quality through the bug-fixing process. Additionally, code review positively contributes to performance, as LLMs may perform static code analysis and identify bugs.

Impact of Code Dependency Graph The Code Dependency Graph, G, plays a vital role in AGILECODER, as demonstrated by the results in Table 4. The variant of AGILECODER without the graph G is prone to exceeding context length errors, leading to early and unexpected termination, while AGILECODER itself does not encounter this issue. This is the primary reason why the version without G appears to run faster. Notably, the presence of G leads to a substantial improvement in executability, increasing from 23.38% to 59.03%. The absence of the graph G can result in a random order among testing scripts, causing inconsistencies during the bug-fixing process. Moreover, ignoring G means that all source code is always included in the instructions, potentially overwhelming and even harming LLMs due to irrelevant information and increasing costs.

Statistical Index	AGILECODER	w/o G
Executability	**59.03**	28.901
Running Time (s)	402	**386**
Token Usage	**29158**	37372
Expenses (USD)	**0.37**	0.39
#Errors	**0**	18
#ExceedingCL	**0**	20

Table 4: Ablation study on the impact of G. #ExceedingCL is the total number of Exceeding Context Length. In the case of the lack of G, we only consider tasks that do not encounter the Exceeding Context Length issue.

5 Conclusion

In this paper, we introduce AGILECODER, a novel multi-agent software development framework inspired by Agile Methodology, promoting dynamic adaptability and iterative development. A key innovation, the Dynamic Code Graph Generator (DCGG), creates a Code Dependency Graph (CDG) to track evolving code relationships. Extensive evaluations on established benchmarks like HumanEval, MBPP, and our newly proposed benchmark, ProjectDev, confirm that AGILECODER achieves new state-of-the-art performance, outperforming recent SOTA methods such as MetaGPT and ChatDev. The success of AGILECODER highlights the potential of integrating Agile Methodology and static analysis techniques into multi-agent software development frameworks. In conclusion, AGILECODER showcases the synergy of Agile Methodology, multi-agent systems, and static analysis, representing a significant advancement in software development automation.

Limitations

While AGILECODER has demonstrated significant advancements in multi-agent software development, there are several areas for future work and limitations to be addressed. One potential avenue for future research is the incorporation of additional Agile practices into the framework. For example, integrating pair programming or continuous integration and deployment (CI/CD) techniques could further enhance the collaboration and efficiency of the multi-agent system. Exploring the adaptation of other Agile methodologies, such as Kanban or Lean, could also provide valuable insights and improvements to the framework.

Another area for future work is the extension of AGILECODER to domains beyond software development. The principles of Agile Methodology and the multi-agent approach could be applied to other complex, iterative tasks such as product design, project management, or scientific research. Investigating the generalizability of the framework to these domains could lead to novel applications and advancements in various fields.

However, AGILECODER also has some limitations that should be acknowledged. One limitation is the reliance on LLMs for code generation and decision-making. While the integration of static analysis through the DCGG helps to mitigate some of the limitations of LLMs, there may still be cases where the generated code is suboptimal or fails to fully meet the requirements. Further research into improving the robustness and reliability of LLM-based code generation could help address this limitation.

Another limitation is the potential scalability issues when dealing with large, complex software projects. As the codebase grows, the computational resources required to maintain and update the CDG may become prohibitive. Future work could explore optimizations to the DCGG or alternative approaches to context retrieval that can handle larger-scale projects more efficiently. Finally, the current implementation of AGILECODER focuses primarily on the technical aspects of software development, such as code generation and testing. However, real-world Agile development also involves important non-technical factors, such as team dynamics, communication, and project management. Incorporating these aspects into the multi-agent framework could provide a more comprehensive and realistic simulation of Agile software development. Despite these limitations, AGILECODER represents a significant step forward in the automation of software development using multi-agent systems and Agile Methodology. By addressing these limitations and exploring the potential for future work, researchers can continue to push the boundaries of what is possible in this exciting field.

References

2024. An agile guide to the planning processes. https://www.pmi.org/learning/library/agile-guide-planning-agile-approach-6837. Accessed: date-of-access.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Lakshya Agrawal, Aditya Kanade, Navin Goyal, Shuvendu K Lahiri, and Sriram Rajamani. 2023. Monitor-guided decoding of code LMs with static analysis of repository context. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Loubna Ben Allal, Raymond Li, Denis Kocetkov, Chenghao Mou, Christopher Akiki, Carlos Munoz Ferrandis, Niklas Muennighoff, Mayank Mishra, Alex Gu, Manan Dey, et al. 2023. Santacoder: don't reach for the stars! *arXiv preprint arXiv:2301.03988*.

Miltiadis Allamanis, Earl T Barr, Premkumar Devanbu, and Charles Sutton. 2018. A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR)*, 51(4):1–37.

AI Anthropic. 2024. Introducing the next generation of claude.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021a. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021b. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Ramakrishna Bairi, Atharv Sonwane, Aditya Kanade, Arun Iyer, Suresh Parthasarathy, Sriram Rajamani, B Ashok, Shashank Shet, et al. 2023. Codeplan: Repository-level coding using llms and planning. *arXiv preprint arXiv:2309.12499*.

Matej Balog, Alexander L Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. 2016. Deepcoder: Learning to write programs. *arXiv preprint arXiv:1611.01989*.

Nghi DQ Bui and Lingxiao Jiang. 2018. Hierarchical learning of cross-language mappings through distributed vector representations for code. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, pages 33–36.

Nghi DQ Bui, Hung Le, Yue Wang, Junnan Li, Akhilesh Deepak Gotmare, and Steven CH Hoi. 2023. Codetf: One-stop transformer library for state-of-the-art code llm. *arXiv preprint arXiv:2306.00029*.

Nghi DQ Bui, Yue Wang, and Steven Hoi. 2022. Detect-localize-repair: A unified framework for learning to debug with codet5. *arXiv preprint arXiv:2211.14875*.

Nghi DQ Bui, Yijun Yu, and Lingxiao Jiang. 2021. Treecaps: Tree-based capsule networks for source code processing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 30–38.

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

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

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. Unixcoder: Unified cross-modal pre-training for code representation. *arXiv preprint arXiv:2203.03850*.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. 2020. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y Wu, YK Li, et al. 2024a. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. 2024b. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. 2021. Measuring coding challenge competence with apps (2021). *arXiv preprint arXiv:2105.09938*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta programming for multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Dong Huang, Qingwen Bu, Jie M Zhang, Michael Luck, and Heming Cui. 2023. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. 2023. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.

Feng Lin, Dong Jae Kim, et al. 2024. When llm-based code generation meets the software development process. *arXiv preprint arXiv:2403.15852*.

Tianyang Liu, Canwen Xu, and Julian McAuley. 2024. Repobench: Benchmarking repository-level code auto-completion systems. In *The Twelfth International Conference on Learning Representations*.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. 2024. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*.

Qinyu Luo, Yining Ye, Shihao Liang, Zhong Zhang, Yujia Qin, Yaxi Lu, Yesai Wu, Xin Cong, Yankai Lin, Yingli Zhang, et al. 2024a. Repoagent: An llm-powered open-source framework for repository-level code documentation generation. *arXiv preprint arXiv:2402.16667*.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2024b. Wizardcoder: Empowering code large language models with evol-instruct. In *The Twelfth International Conference on Learning Representations*.

Potsawee Manakul, Adian Liusie, and Mark Gales. 2023. SelfCheckGPT: Zero-resource black-box hallucination detection for generative large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9004–9017, Singapore. Association for Computational Linguistics.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*.

Huy N Phan, Hoang N Phan, Tien N Nguyen, and Nghi DQ Bui. 2024. Repohyper: Better context retrieval is all you need for repository-level code completion. *arXiv preprint arXiv:2403.06095*.

Nikhil Pinnaparaju, Reshinth Adithyan, Duy Phung, Jonathan Tow, James Baicoianu, Ashish Datta, Maksym Zhuravinskyi, Dakota Mahan, Marco Bellagente, Carlos Riquelme, et al. 2024. Stable code technical report. *arXiv preprint arXiv:2404.01226*.

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

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Disha Shrivastava, Denis Kocetkov, Harm de Vries, Dzmitry Bahdanau, and Torsten Scholak. 2023a. Repofusion: Training code models to understand your repository. *arXiv preprint arXiv:2306.10998*.

Disha Shrivastava, Hugo Larochelle, and Daniel Tarlow. 2023b. Repository-level prompt generation for large language models of code. In *International Conference on Machine Learning*, pages 31693–31715. PMLR.

Daniel Tang, Zhenghan Chen, Kisub Kim, Yewei Song, Haoye Tian, Saad Ezzini, Yongfeng Huang, and Jacques Klein Tegawende F Bissyande. 2024. Collaborative agents for software engineering. *arXiv preprint arXiv:2402.02172*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi DQ Bui, Junnan Li, and Steven CH Hoi. 2023. Codet5+: Open code large language models for code understanding and generation. *arXiv preprint arXiv:2305.07922*.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859*.

Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, pages 1–10.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*.

Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. 2024. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. *arXiv preprint arXiv:2401.07339*.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, et al. 2023. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x. *arXiv preprint arXiv:2303.17568*.

Tianyu Zheng, Ge Zhang, Tianhao Shen, Xueling Liu, Bill Yuchen Lin, Jie Fu, Wenhu Chen, and Xiang Yue. 2024. Opencodeinterpreter: Integrating code generation with execution and refinement. *arXiv preprint arXiv:2402.14658*.

Li Zhong, Zilong Wang, and Jingbo Shang. 2024. Ldb: A large language model debugger via verifying runtime execution step-by-step. *arXiv preprint arXiv:2402.16906*.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2023. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*.

A Appendix

A.1 Experimental Setting

In our evaluation, we use GPT-3.5-Turbo-0613 and claude-3-haiku@20240307, and we set the temperature to 0.2, and top_p to 1, in accordance with previous work (Qian et al., 2023).

HumanEval (Chen et al., 2021) includes 164 handwritten programming problems, MBPP (Austin et al., 2021a) is comprised of 427 Python

tasks, and ProjectDev consists of 30 software development tasks. For each dataset, every sample or task is executed three times, and the average results are reported, except for #Errors and #ExceedingCL, which are presented as total counts.

A.2 More details on Analysis

A.2.1 Impact of The Three-step Code Review

In the Subsection 3.2.2, we introduce a three-step code review to conduct a static check for the correctness of source code. The detailed procedure is as below:

1. **Basic Implementation Checks:** Verification of implementation basics, such as the presence of empty methods (e.g., *pass* in Python), adequacy of docstrings, and completeness of import statements.

2. **Backlog Compliance:** Assessment of whether source code fulfills the tasks listed in the sprint backlog and includes all specified features.

3. **Criteria Satisfaction and Bug Identification:** Final review to ensure the code meets the sprint's acceptance criteria and is free from potential bugs.

We conduct an analysis to demonstrate the impact of our proposed three-step code review by comparing AGILECODER with its variant, where the three steps are condensed into a single step. Results in Table 5 show that breaking the code review into three steps improves accuracy. This is because the multi-step prompting strategy allows LLMs to perform static analyses more effectively and provide more precise feedback. In contrast, a single-step review involves instructions that encompass all desired constraints, making it more challenging for LLMs to understand and follow, thereby diminishing overall effectiveness.

	Model	HumanEval
GPT-3.5 Turbo	AGILECODER	70.53
	w/o three-step review	67.68 (-2.85)
Claude 3 Haiku	AGILECODER	79.27
	w/o three-step review	75.61 (-3.66)

Table 5: Ablation study on our three-step prompting strategy for code review

A.2.2 Error Bar

In addition to main results in the Section 4.1, we also report error bars to demonstrate the robustness of our method, AGILECODER, under different runs. Results in Table 6 show that AGILECODER exhibits only minor variations across different runs.

Model	Dataset	
	HumanEval	MBPP
GPT-3.5 Turbo	70.53± 0.70	80.92± 0.83
Claude 3 Haiku	79.27± 0.86	84.31± 0.34

Table 6: Error bars of AGILECODER on HumanEval and MBPP

A.2.3 Capabilities Analysis

In this subsection, we provide an analysis on capabilities of multi-agent frameworks for software engineering tasks. As presented in Table 7, compared to ChatDev (Qian et al., 2023) and MetaGPT (Hong et al., 2024), AGILECODER has three additional capabilities, including flexible progress, incremental development, and modeling file dependencies. Incorporating these features can enhance the ability to perform software engineering tasks.

Framework Capabiliy	ChatDev	MetaGPT	AGILECODER
Code generation	✓	✓	✓
Role-based task management	✓	✓	✓
Code review	✓	✓	✓
Flexible progress	✗	✗	✓
Incremental development	✗	✗	✓
Modeling file dependencies	✗	✗	✓

Table 7: Comparison with other multi-agent frameworks

A.2.4 Example of Code Dependency Graph

In the Section 3.3, we propose a novel dependency graph G modeling relationships among code files. Figure 4 provides an illustrative example of the graph G with each node representing a code file. The file *library.py* is contingent on the file *book.py*, so there is an edge from *library.py* to *book.py*.

A.3 Details for ProjectDev

In this section, we present our dataset construction, detailed evaluation process, and the ProjectDev. We use GPT-3.5-Turbo-0613 as the backend model for the sake of evaluation.

A.3.1 ProjectDev construction process

To ensure transparency and reproducibility, we meticulously have to curate ProjectDev through the process below.

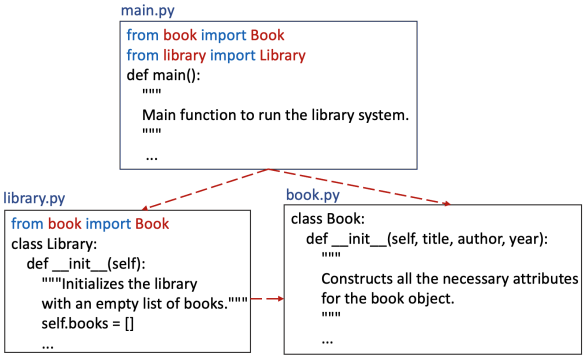

Figure 4: An example of the Code Dependency Graph

- Step 1: We reuse 7 tasks from MetaGPT (Hong et al., 2024) and supplement them with additional software development tasks generated using GPT-4o and from Internet, ensuring diversity across multiple domains.

- Step 2: For each task, GPT-4o list detailed requirements with at least 10 atomic items that can be checked for completion, which we manually review for plausibility and feasibility.

- Step 3: We evaluate each task by running it three times, focusing on practical application through human assessment. If the software fails to run, its executability rate is 0%. Otherwise, we manually interact with the program to measure how many requirements are met. For instance, if a program is executable and meets 4 out of 10 requirements, its executability rate is 40%.

A.3.2 Evaluation Steps

The entire evaluation process is conducted manually to ensure the correctness of the evaluation results. Participants are experienced developers with at least two years of Python programming experience. For each task, we run a method three times using the same prompt, producing a program for each run. For each generated program, we attempt to execute this program. If the program is executable, we evaluate it against all expected requirements. The final score is determined by the percentage of requirements the program meets. Below is the pseudocode for evaluating each task step-by-step:

A.3.3 ProjectDev dataset

Task Id: 1
Task: Snake game

Input: A task t and its corresponding programs produced by ChatDev, MetaGPT, and AgileCoder
Output: Final score as a percentage
Load programs generated from ChatDev, MetaGPT, and AgileCoder that correspond to the task t
Load *requirements* list of the task t from the ProjectDev dataset
Initialize *requirements_met* to 0
Initialize *total_requirements* to length of *requirements* list
Attempt to run each program
if program is executable **then**
  for each requirement r in *requirements* list **do**
    if the requirement r is met **then**
      Increment *requirements_met* by 1
    end if
  end for
end if
Calculate *final_score* as $(requirements_met/total_requirements) \times 100\%$
return *final_score*

Prompt: Create a snake game
Requirements:

- **Game Board:**
  - ◇ Create a grid-based game board.
  - ◇ Define the dimensions of the grid (e.g., 10x10).
  - ◇ Display the grid on the screen.
- **Snake Initialization:**
  - ◇ Place the snake on the game board.
  - ◇ Define the initial length and starting position of the snake.
  - ◇ Choose a direction for the snake to start moving (e.g., right).
- **Snake Movement:**
  - ◇ Implement arrow key controls for snake movement.
  - ◇ Ensure the snake moves continuously in the chosen direction.
  - ◇ Update the snake's position on the grid.
- **Food Generation:**

- ◇ Generate food at random positions on the game board.
- ◇ Ensure food doesn't appear on the snake's body.

- **Collision Handling:**
  - ◇ Detect collisions between the snake and the game board boundaries.
  - ◇ Detect collisions between the snake's head and its body.
  - ◇ Detect collisions between the snake's head and the food.
- **Snake Growth:**
  - ◇ Increase the length of the snake when it consumes food.
  - ◇ Add a new segment to the snake's body.
- **Score Display:**
  - ◇ Implement a scoring system.
  - ◇ Display the current score on the screen.
- **Game Over Condition:**
  - ◇ Trigger a game over scenario when the snake collides with the boundaries.
  - ◇ Trigger a game over scenario when the snake collides with its own body.
  - ◇ Display a game over message.
  - ◇ Allow the player to restart the game.
- **Graphics and User Interface:**
  - ◇ Use graphics or ASCII characters to represent the snake and food.
  - ◇ Design a user-friendly interface with clear instructions and score display.
- **Animations and Effects:**
  - ◇ Add animations for snake movement and growth.
  - ◇ Implement visual effects for collisions and food consumption.

Task Id: 2
Task: Brick breaker game
Prompt: Create a brick breaker game
Requirements:

- **Game Board:**
  - ◇ Create a game board with a grid-based layout.
  - ◇ Define the dimensions of the game board.
  - ◇ Display the game board on the screen.
- **Paddle Setup:**
  - ◇ Add a paddle at the bottom of the screen.
  - ◇ Allow the player to control the paddle using keyboard or touch controls.
- **Brick Formation:**
  - ◇ Generate a formation of bricks on the top of the screen.
  - ◇ Define the number of rows and columns of bricks.
  - ◇ Assign different colors or types to bricks.
- **Ball Initialization:**
  - ◇ Place a ball on the paddle at the beginning of the game.
  - ◇ Enable the player to launch the ball.
- **Ball Movement:**
  - ◇ Implement physics for the ball's movement.
  - ◇ Allow the ball to bounce off walls, paddle, and bricks.
  - ◇ Update the ball's position continuously.
- **Collision Detection:**
  - ◇ Detect collisions between the ball and the paddle.
  - ◇ Detect collisions between the ball and the bricks.
  - ◇ Handle different types of collisions appropriately.
- **Brick Destruction:**
  - ◇ Remove a brick when the ball collides with it.
  - ◇ Implement different points for different brick types.
  - ◇ Track the number of bricks remaining.
- **Scoring System:**
  - ◇ Implement a scoring system based on the bricks destroyed.
  - ◇ Display the current score on the screen.
- **Power-ups:**

- Introduce power-ups that fall when certain bricks are destroyed.
- Implement power-ups such as extra balls, larger paddle, or slower ball speed.

- **Game Levels:**
  - Create multiple levels with increasing difficulty.
  - Design new brick formations for each level.

- **Game Over Condition:**
  - End the game when the ball goes below the paddle.
  - Display a game over message.
  - Allow the player to restart the game.

- **Graphics and User Interface:**
  - Use graphics to represent bricks, paddle, and ball.
  - Design a user-friendly interface with clear instructions and score display.

Task Id: 3
Task: 2048 game
Prompt: Create a 2048 game
Requirements:

- **Game Board:**
  - Create a 4x4 grid as the game board.
  - Display the grid on the web page.

- **Tile Generation:**
  - Generate two initial tiles with the values 2 or 4 at random positions on the grid.
  - Update the display to show the initial tiles.

- **Tile Movement:**
  - Implement arrow key controls for tile movement (up, down, left, right).
  - Allow tiles to slide in the chosen direction until they encounter the grid boundary or another tile.
  - Combine tiles with the same value when they collide.

- **Score Tracking:**
  - Implement a scoring system.
  - Display the current score on the web page.
  - Update the score when tiles are combined.

- **Winning Condition:**
  - Define the winning condition as reaching the 2048 tile value.
  - Display a victory message when the player reaches the winning condition.
  - Allow the player to continue playing after winning.

- **Game Over Condition:**
  - Implement a game over scenario when there are no more valid moves.
  - Display a game over message.
  - Allow the player to restart the game.

- **Animations and Transitions:**
  - Add smooth animations for tile movements.
  - Implement transitions for tile merging.

- **Graphics and Styling:**
  - Use graphics or stylized numbers to represent different tile values.

Task Id: 4
Task: Flappy bird game
Prompt: Write code for Flappy Bird in python where you control a yellow bird continuously flying between a series of green pipes. The bird flaps every time you left click the mouse. If it falls to the ground or hits a pipe, you lose. This game goes on indefinitely until you lose; you get points the further you go.
Requirements:

- **Game Elements:**
  - Create a yellow bird as the main character.
  - Design green pipes for the bird to navigate through.
  - Set up a ground or background for the game.

- **Bird Movement:**
  - Implement continuous forward movement of the bird.

- ◇ Make the bird fall due to gravity.
- ◇ Allow the bird to jump or "flap" when the left mouse button is clicked.

- **Pipe Generation:**
  - ◇ Generate a series of pipes at regular intervals.
  - ◇ Randomize the height of the gaps between pipes.
  - ◇ Remove pipes from the screen when they move off the left side.
- **Collision Detection:**
  - ◇ Detect collisions between the bird and the ground.
  - ◇ Detect collisions between the bird and the pipes.
- **Score Tracking:**
  - ◇ Implement a scoring system based on the distance traveled.
  - ◇ Display the current score on the screen.
- **Game Over Condition:**
  - ◇ Trigger a game over scenario when the bird hits the ground.
  - ◇ Trigger a game over scenario when the bird hits a pipe.
  - ◇ Display a game over message.
  - ◇ Allow the player to restart the game.
- **Animations and Effects:**
  - ◇ Add animations for bird flapping, pipe movement, and game over transitions.
  - ◇ Implement visual effects for collisions.

Task Id: 5
Task: Tank battle game
Prompt: Create a tank battle game
Requirements:

- **Game Board:**
  - ◇ Create a grid-based game board representing the battlefield.
  - ◇ Define the dimensions of the grid.
  - ◇ Display the grid on the screen.
- **Tank Initialization:**
  - ◇ Place two tanks on the game board for a two-player game.
  - ◇ Set initial positions for each tank.
  - ◇ Allow players to control their tanks using keyboard controls.
- **Obstacles:**
  - ◇ Generate obstacles (e.g., walls, barriers) on the game board.
  - ◇ Ensure obstacles are placed strategically to create a challenging battlefield.
- **Tank Movement:**
  - ◇ Implement controls for tank movement (e.g., forward, backward, rotate left, rotate right).
  - ◇ Allow tanks to move freely on the grid.
  - ◇ Restrict tank movement when colliding with obstacles.
- **Tank Firing:**
  - ◇ Implement controls for firing projectiles (e.g., bullets, missiles).
  - ◇ Limit the firing rate to prevent spamming.
  - ◇ Display projectiles on the screen.
- **Projectile Collision:**
  - ◇ Detect collisions between projectiles and tanks.
  - ◇ Deal damage to tanks when hit by projectiles.
  - ◇ Remove projectiles when they collide with obstacles or go off the screen.
- **Health and Damage:**
  - ◇ Assign health values to tanks.
  - ◇ Display health bars for each tank.
  - ◇ Trigger explosions or visual effects when a tank is destroyed.
- **Scoring System:**
  - ◇ Implement a scoring system based on the number of tanks destroyed.
  - ◇ Display the current score on the screen.
- **Game Over Condition:**
  - ◇ Trigger a game over scenario when a tank's health reaches zero.
  - ◇ Display a game over message.
  - ◇ Allow players to restart the game.

- **Graphics and User Interface:**
  - ◇ Use graphics to represent tanks, projectiles, and obstacles.
  - ◇ Design a user-friendly interface with clear instructions and score display.
- **Animations and Effects:**
  - ◇ Add animations for tank movement, firing, and explosions.
  - ◇ Implement visual effects for collisions and explosions.

Task Id: 6
Task: Excel data process
Prompt: Write an excel data processing program based on streamlit and pandas. The screen first shows an excel file upload button. After the excel file is uploaded, use pandas to display its data content. The program is required to be concise, easy to maintain, and not over-designed. It uses streamlit to process web screen displays, and pandas is sufficient to process excel reading and display. Please make sure others can execute directly without introducing additional packages.
Requirements:

- **File Upload Button:**
  - ◇ Display a file upload button using any web library (such as Streamlit).
  - ◇ Allow users to upload Excel files.
- **Pandas Data Processing:**
  - ◇ Use Pandas to read the uploaded Excel file.
  - ◇ Load the data into a Pandas DataFrame.
- **Data Display:**
  - ◇ Display the content of the DataFrame.
  - ◇ Show the first few rows of the data by default.
- **Toggle for Full Data Display:**
  - ◇ Add a toggle button to switch between displaying the first few rows and the full data.
- **Handling Missing Values:**
  - ◇ Check for and handle missing values in the data.
- **Column Selection:**
  - ◇ Allow users to select specific columns for display.
  - ◇ Display only the selected columns.
- **Filtering:**
  - ◇ Implement simple data filtering based on user input.
  - ◇ Display the filtered results.
- **Sorting:**
  - ◇ Allow users to sort the data based on one or more columns.
  - ◇ Display the sorted results.
- **Download Processed Data:**
  - ◇ Provide a button to allow users to download the processed data.
- **Visualizations (Optional):**
  - ◇ Include optional simple visualizations (e.g., bar chart, line chart).
- **Error Handling:**
  - ◇ Implement error handling for file upload issues or data processing errors (e.g., file is not in excel or csv format).
  - ◇ Display informative messages to the user.

Task Id: 7
Task: CRUD manage
Prompt: Write a management program based on the CRUD addition, deletion, modification and query processing of the customer business entity. The customer needs to save this information: name, birthday, age, sex, and phone. The data is stored in client.db, and there is a judgement whether the customer table exists. If it doesn't, it needs to be created first. Querying is done by name; same for deleting. The program is required to be concise, easy to maintain, and not over-designed. The screen is realized through streamlit and sqlite—no need to introduce other additional packages.
Requirements:

- **Database Initialization:**
  - ◇ Connect to the SQLite database (client.db).

- ◇ Check if the customer table exists.
- ◇ If not, create the customer table with fields: name, birthday, age, sex, and phone.

- **Add Customer (Create):**
  - ◇ Provide input fields for name, birthday, age, sex, and phone.
  - ◇ Allow users to add a new customer to the database.
  - ◇ Validate input data (e.g., check if the phone is valid).
- **Query Customer (Read):**
  - ◇ Implement a query interface with an input field for the customer's name.
  - ◇ Display the customer information if found.
  - ◇ Provide a message if the customer is not found.
- **Update Customer (Modify):**
  - ◇ Allow users to update customer information.
  - ◇ Display the current information and provide input fields for modifications.
  - ◇ Update the database with the modified data.
- **Delete Customer (Delete):**
  - ◇ Allow users to delete a customer based on their name.
  - ◇ Display a confirmation message before deleting.
  - ◇ Update the database by removing the customer.
- **Display All Customers:**
  - ◇ Create a section to display all customers in the database.
  - ◇ Display relevant information for each customer.
- **UI:**
  - ◇ Design a (Streamlit) user interface with a clean layout.
  - ◇ Use (Streamlit) components for input fields, buttons, and information display.
- **Error Handling:**
  - ◇ Implement error handling for database connectivity issues.
  - ◇ Provide user-friendly error messages.

Task Id: 8
Task: Custom press releases
Prompt: Create custom press releases; develop a Python script that extracts relevant information about company news from external sources, such as social media; extract update interval database for recent changes. The program should create press releases with customizable options and export writings to PDFs, NYTimes API JSONs, media format styled with interlink internal fixed character-length metadata.
Requirements:

- **Data Extraction from External Sources:**
  - ◇ Implement web scraping or use APIs to extract relevant information from social media and other external sources.
  - ◇ Extract data such as company updates, news, and events.
- **Update Interval Database:**
  - ◇ Develop a database to store information about recent changes or updates.
  - ◇ Include fields like timestamp, source, and content.
- **Customizable Press Release Options:**
  - ◇ Design a user interface or command-line options for users to customize press releases.
  - ◇ Allow customization of content, format, and metadata.
- **Press Release Content Generation:**
  - ◇ Develop algorithms to generate coherent and concise press release content.
  - ◇ Use extracted information to create engaging narratives.
- **Export Options:**
  - ◇ Provide options to export press releases in different formats, such as PDF and NYTimes API JSONs.
  - ◇ Include customizable templates for different media formats.
- **Metadata Inclusion:**

- ◇ Add metadata to the press releases, including fixed character-length metadata.
- ◇ Ensure metadata includes relevant information such as publication date, source, and author.

- **PDF Export:**
  - ◇ Implement functionality to export press releases as PDF files.
  - ◇ Allow users to specify PDF export options (e.g., layout, fonts).

- **NYTimes API JSON Export:**
  - ◇ Integrate with the NYTimes API to fetch additional relevant information.
  - ◇ Format and export the press releases in JSON format compatible with the NYTimes API.

- **Media Format Styling:**
  - ◇ Apply styling to the press releases based on different media formats.
  - ◇ Ensure the styling is consistent with industry standards.

- **Interlink Internal Content:**
  - ◇ Implement interlinking of internal content within the press releases.
  - ◇ Include links to relevant articles, documents, or resources.

- **Error Handling:**
  - ◇ Implement error handling mechanisms for data extraction, customization, and export processes.
  - ◇ Provide clear error messages and logging.

Task Id: 9
Task: Caro game
Prompt: Create a caro game in python
Requirements:

- **Game Board:**
  - ◇ Create a grid-based game board for Caro.
  - ◇ Define the dimensions of the board (commonly 15x15 for Caro).
  - ◇ Display the game board on the console or a graphical user interface.

- **Player vs. Player:**
  - ◇ Implement a two-player mode where two human players can take turns.
  - ◇ Allow players to place their marks (X or O) on the board.

- **Winning Conditions:**
  - ◇ Detect and announce a winner when a player gets five marks in a row horizontally, vertically, or diagonally.
  - ◇ Declare a draw when the board is full and no player has won.

- **User Interface:**
  - ◇ Create a user-friendly interface for players to interact with the game.
  - ◇ Display the current state of the board after each move.

- **Input Handling:**
  - ◇ Implement input handling for player moves.
  - ◇ Ensure valid moves and handle invalid inputs gracefully.

- **Restart and Exit Options:**
  - ◇ Provide options to restart the game or exit the program after a game is completed.
  - ◇ Ask for confirmation before restarting or exiting.

- **Game Logic:**
  - ◇ Implement the core game logic, including checking for winning conditions, updating the board, and managing turns.

Task Id: 10
Task: Video player
Prompt: Create a video player in python that can play videos from users. The application can also support multiple controls such as play, pause and stop.
Requirements:

- **User Interface:**
  - ◇ Develop a user interface to display the video player.
  - ◇ Include a section to display the video content.
  - ◇ Create a space for control buttons (play, pause, stop).

- **Video Loading:**
  - ◇ Implement functionality to load videos from user input.
  - ◇ Support common video formats (e.g., MP4, AVI).
  - ◇ Handle errors gracefully if the video format is not supported.
- **Play Button:**
  - ◇ Implement a "Play" button to start playing the loaded video.
  - ◇ Ensure the button is responsive and updates its state (e.g., changes to a pause button when the video is playing).
- **Pause Button:**
  - ◇ Implement a "Pause" button to temporarily pause the video.
  - ◇ Allow users to resume playback from the paused state.
- **Stop Button:**
  - ◇ Implement a "Stop" button to stop the video playback.
  - ◇ Reset the video to the beginning when stopped.
- **Time Slider:**
  - ◇ Add a time slider or progress bar to visualize the current position in the video.
  - ◇ Allow users to click on the slider to jump to specific points in the video.
- **Error Handling:**
  - ◇ Implement error handling for cases where the video fails to load or encounters playback issues.
  - ◇ Display informative error messages to users.

Task Id: 11
Task: Youtube Video Downloader
Prompt: Create a Youtube Video Downloader in Python that receives a Youtube link as an input and then downloads the video with multiple options of resolutions
Requirements:

- **User Input:**
  - ◇ Develop a user interface or command-line interface to accept a YouTube video link as input.
- **YouTube API Integration (Optional):**
  - ◇ Integrate with the YouTube API to fetch information about the available video resolutions and formats.
  - ◇ Retrieve details such as video title, available resolutions, and formats.
- **Video Resolution Options:**
  - ◇ Present the user with options to choose from various video resolutions.
  - ◇ Display information about each resolution (e.g., resolution, format, file size).
- **Download Mechanism:**
  - ◇ Implement the video download mechanism using a library like pytube or similar.
  - ◇ Allow users to choose the desired video resolution.
- **Download Progress Display:**
  - ◇ Display a progress bar or percentage to show the download progress.
  - ◇ Update the progress in real-time during the download.
- **File Naming Options:**
  - ◇ Provide options for users to specify the name of the downloaded file.
  - ◇ Generate a default name based on the video title.
- **Download Location:**
  - ◇ Allow users to specify the directory where the video will be saved.
  - ◇ Use a default directory if the user doesn't specify one.
- **Video Information Display:**
  - ◇ Display relevant information about the video, such as title, duration, and uploader.
  - ◇ Show information before and after the download.
- **Error Handling:**

- Implement error handling to gracefully handle issues such as invalid URLs, network errors, or download failures.

Task Id: 12
Task: QR Code Generator and Detector
Prompt: Create a Python program that produces a QR code for the input from users and decodes a QR code from users.
Requirements:

- **Generating QR Codes:**
  - **User Input:**
    - Develop a user interface or command-line interface to accept user input.
    - Allow users to input text or a URL for which a QR code will be generated.
  - **QR Code Generation Library:**
    - Choose a QR code generation library in Python (e.g., qrcode).
  - **QR Code Generation:**
    - Generate QR code successfully with valid input.
    - Allow users to customize QR code parameters (e.g., size, color).
  - **Display QR Code:**
    - Display the generated QR code to the user on screen.
  - **Save QR Code:**
    - Implement an option for users to save the generated QR code as an image file (e.g., PNG).
- **Decoding QR Codes:**
  - **User Input for Decoding:**
    - Allow users to input an image file containing a QR code for decoding.
    - Support common image formats (e.g., PNG, JPEG).
  - **QR Code Decoding Library:**
    - Choose a QR code decoding library in Python (e.g., opencv, pyzbar).
  - **QR Code Decoding:**
    - Decode QR code successfully with valid input.
    - Display the decoded text or URL to the user.
  - **Error Handling:**
    - Implement error handling for cases where decoding fails or the input is invalid.

Task Id: 13
Task: To-Do List App
Prompt: Create a simple to-do list application in python where users can add, edit, and delete tasks. The application includes some features like marking tasks as completed and categorizing tasks.
Requirements:

- **User Interface:**
  - Provide options for adding, editing, deleting, marking as completed, and categorizing tasks.
- **Task List Display:**
  - Display the list of tasks in a readable format.
  - Include information such as task name, status (completed or not), and category.
- **Adding Tasks:**
  - Implement functionality to add new tasks to the to-do list.
  - Allow users to input the task name, status, and category.
- **Editing Tasks:**
  - Provide an interface to modify task details such as name, status, and category.
- **Deleting Tasks:**
  - Implement the ability to delete tasks from the to-do list.
  - Confirm user intent before deleting a task.
- **Marking Tasks as Completed:**
  - Allow users to mark tasks as completed.
  - Toggle their completion status.
- **Categorizing Tasks:**
  - Filter tasks by category.
- **Saving and Loading Tasks:**
  - Save the to-do list data to a file (e.g., JSON or CSV).

- Load the saved data when the application starts.

Task Id: 14
Task: Calculator
Prompt: Create a calculator with python that performs basic arithmetic operations. Practices user input, functions, and mathematical operations.
Requirements:

- **User Interface:**
  - Create an interface for basic arithmetic operations (addition, subtraction, multiplication, division).
- **User Input:**
  - Accept user input for numeric values and mathematical operations.
  - Allow multiple input numbers and operations.
- **Arithmetic Operations:**
  - Successfully implement functions for addition operations.
  - Successfully implement functions for subtraction operations.
  - Successfully implement functions for multiplication operations.
  - Successfully implement functions for division operations.
- **Functionality:**
  - Allow users to choose the operation they want to perform.
- **Calculation Execution:**
  - Display the result of the calculation.
- **Continuous Calculation (Optional):**
  - Optionally, allow users to perform continuous calculations without restarting the program.
- **Error Handling:**
  - Implement error handling for cases where the user provides invalid input or attempts an unsupported operation.

Task Id: 15
Task: Chatbot
Prompt: Create a Streamlit chatbot powered by LLMs on Hugging Face by following the requirements below:
1) Download `Qwen/Qwen2-7B-Instruct` and load this model as the backend model
2) The website has elements:

- **Input Box:** For user input.
- **Output Display:** To show the chatbot's responses.
- **Sidebar:** To control the generation parameters, such as temperature and top_k.

Requirements:

- **Model Setup:**
  - Download the `'Qwen/Qwen2-7B-Instruct'` model from Hugging Face.
  - Load the model as the backend for the chatbot.
- **User Interface:**
  - Create an input box using Streamlit for user queries.
  - Display the chatbot's responses in the main area of the web app.
  - Implement a sidebar in Streamlit for additional controls.
- **Sidebar Controls:**
  - Provide sliders or input fields to adjust generation parameters like temperature and top_k.
  - Allow users to customize settings for the chatbot's response behavior.
- **Chat History:**
  - Maintain a history of the conversation within the session.
  - Display past interactions between the user and the chatbot.
- **User Experience:**
  - Ensure the interface is user-friendly and responsive.

- Allow users to clear the chat history if desired.

- **Error Handling:**
  - Implement error handling for issues like model loading failures or invalid inputs.
  - Display informative error messages to guide the user.

Task Id: 16
Task: Number Guessing Game
Prompt: Create a number guessing game in Python. This is a text-based game where the computer selects a random number, and the user has to guess it within a certain number of tries, receiving feedback if the guess is too high or too low.
Requirements:

- **Game Initialization:**
  - Have the computer select a random number within a specified range (e.g., 1 to 100).
  - Set a maximum number of attempts for the player to guess the number.

- **User Interaction:**
  - Prompt the user to input their guess.
  - Provide feedback if the guess is too high.
  - Provide feedback if the guess is too low.
  - Inform the user if they've guessed the correct number.

- **Attempt Tracking:**
  - Keep track of the number of guesses the user has made.
  - Inform the user of remaining attempts after each guess.

- **Game Over Conditions:**
  - End the game when the user guesses correctly.
  - End the game when the user runs out of attempts and reveal the correct number.

- **Replay Option:**
  - Offer the user the option to play again after the game ends.
  - Reset the game state if the user chooses to replay.

Task Id: 17
Task: Pong Game
Prompt: Create a Pong game in Python. This is a simple two-player paddle game where each player controls a paddle to bounce a ball back and forth. The objective is to prevent the ball from passing your paddle.
Requirements:

- **Game Window:**
  - Create a game window with appropriate dimensions (e.g., 800x600 pixels).
  - Set up the game's frame rate and main loop.

- **Paddle Implementation:**
  - Create two paddles, one for each player, positioned on opposite sides of the screen.
  - Allow Player 1 to control their paddle using specific keys (e.g., 'W' and 'S').
  - Allow Player 2 to control their paddle using specific keys (e.g., Up and Down arrows).

- **Ball Mechanics:**
  - Initialize the ball in the center of the screen at game start.
  - Implement ball movement with an initial velocity.
  - Allow the ball to bounce off the top and bottom edges of the window.

- **Collision Detection:**
  - Detect collisions between the ball and the paddles.
  - Adjust the ball's trajectory upon collision with a paddle.

- **Scoring System:**
  - Award a point to a player when the opponent misses the ball.
  - Display the current score for both players on the screen.

- **Game Over Conditions:**
  - Set a winning score (e.g., first to 10 points wins).
  - Display a winning message when a player reaches the winning score.

- **Game Reset:**
  - ◇ Reset the ball to the center after a point is scored.
  - ◇ Allow players to restart the game after it ends.

Task Id: 18
Task: Random Password Generator
Prompt: Create an app in Python that generates secure and random passwords.
Requirements:

- **User Interface:**
  - ◇ Provide a simple interface for users to generate passwords.
  - ◇ Allow users to specify the desired password length.
- **Character Options:**
  - ◇ Offer options to include lowercase letters.
  - ◇ Offer options to include uppercase letters.
  - ◇ Offer options to include numbers.
  - ◇ Offer options to include special characters.
- **Password Generation:**
  - ◇ Implement a function to generate a random password based on selected options.
  - ◇ Ensure the password is generated using secure randomization methods.
- **Display and Usage:**
  - ◇ Display the generated password to the user.
  - ◇ Provide an option to copy the password to the clipboard.

Task Id: 19
Task: Alarm Clock
Prompt: Create an alarm app in Python where users can set an alarm at a specific time. The app will trigger a sound or notification when the time is reached.
Requirements:

- **User Interface:**
  - ◇ Provide input fields for setting the alarm time (hours, minutes, seconds).
  - ◇ Allow users to set the alarm in 12-hour or 24-hour format.
- **Alarm Scheduling:**
  - ◇ Implement functionality to compare the current time with the alarm time.
  - ◇ Schedule the alarm to trigger at the specified time.
- **Notification Mechanism:**
  - ◇ Play a sound when the alarm time is reached.
  - ◇ Display a notification message to the user.
- **Alarm Management:**
  - ◇ Allow users to view, edit, or delete set alarms.
  - ◇ Support multiple alarms being set simultaneously.
- **Error Handling:**
  - ◇ Validate user input to ensure it's a valid time.
  - ◇ Handle exceptions gracefully and inform the user of any issues.

Task Id: 20
Task: Countdown Timer
Prompt: Create a countdown timer in Python where users can set a countdown, and it will notify them when the time is up.
Requirements:

- **User Interface:**
  - ◇ Provide input fields for hours, minutes, and seconds to set the countdown duration.
  - ◇ Display the countdown timer on the screen.
- **Timer Functionality:**
  - ◇ Implement the countdown logic that updates every second.
  - ◇ Display the remaining time in a clear format (HH:MM).
- **Notification Mechanism:**
  - ◇ Play a sound or display a message when the countdown reaches zero.

- ◇ Ensure the notification is noticeable even if the user is not actively viewing the app.

- **Start/Pause/Reset Controls:**
  - ◇ Include buttons to start, pause, and reset the countdown.
  - ◇ Allow users to pause and resume the timer as needed.

- **Error Handling:**
  - ◇ Validate that the input time is a positive number.
  - ◇ Handle incorrect inputs gracefully with appropriate messages.

Task Id: 21
Task: Markdown Viewer
Prompt: Create a Markdown viewer app in Python that allows users to write or open Markdown files and renders them as formatted text in a separate pane.
Requirements:

- **User Interface:**
  - ◇ Design a split-view interface with an editor pane and a preview pane.
  - ◇ Include menu options or buttons for opening and saving files.

- **Markdown Editing:**
  - ◇ Provide a text editor where users can write or paste Markdown content.
  - ◇ Support basic text editing features like undo, redo, and text selection.

- **Real-Time Rendering:**
  - ◇ Implement live rendering of the Markdown content in the preview pane.
  - ◇ Ensure that changes in the editor are immediately reflected in the preview.

- **File Operations:**
  - ◇ Allow users to open existing Markdown files (.md) from their system.
  - ◇ Enable users to save their Markdown content to a file.

- **Supported Markdown Features:**
  - ◇ Render headings, paragraphs, lists, links, and images correctly.
  - ◇ Support code blocks, inline code, blockquotes, and tables.

Task Id: 22
Task: PDF Merger and Splitter
Prompt: Create a Python Streamlit app that can merge multiple PDF files into one and split a PDF into multiple files.
Requirements:

- **PDF Merging:**
  - ◇ Allow users to select multiple PDF files to merge.
  - ◇ Provide options to reorder files before merging.
  - ◇ Merge the selected PDFs into a single file.

- **PDF Splitting:**
  - ◇ Allow users to select a PDF file to split.
  - ◇ Provide options to specify page ranges for splitting.
  - ◇ Generate new PDF files based on specified ranges.

- **User Interface:**
  - ◇ Develop a GUI with file selection dialogs.
  - ◇ Include progress indicators for operations.

- **Error Handling:**
  - ◇ Handle cases where files are corrupted or unreadable.
  - ◇ Inform the user of any issues during processing.

Task Id: 23
Task: Music Player
Prompt: Create a music player app where users can load music files like MP3 and play, pause, or stop them.
Requirements:

- **User Interface:**
  - ◇ Design a clean interface with playback controls (play, pause, stop).
  - ◇ Include a playlist area to display loaded tracks.

- **Music Loading:**

- ◇ Implement functionality to browse and select music files from the user's system.
- ◇ Support common audio file formats such as MP3 and WAV.

- **Playback Controls:**
  - ◇ Enable the play button to start music playback.
  - ◇ Implement pause functionality to halt playback without resetting the track.
  - ◇ Use the stop button to halt playback and reset the track to the beginning.

- **Playlist Management:**
  - ◇ Allow users to add multiple tracks to a playlist.
  - ◇ Provide options to remove tracks from the playlist.

- **Volume and Seek Controls:**
  - ◇ Include a volume slider to adjust playback volume.
  - ◇ Provide a progress bar to display track progress and allow seeking.

- **Track Information:**
  - ◇ Display the current track's metadata (title, artist, album).
  - ◇ Show album art if available.

- **Error Handling:**
  - ◇ Handle unsupported file formats gracefully.
  - ◇ Notify the user of any playback errors.

Task Id: 24
Task: Sudoku Game
Prompt: Create a Sudoku game with a GUI using a library like Tkinter or PyQt.
Requirements:

- **Game Interface:**
  - ◇ Develop a graphical user interface (GUI) for the game using a library like Tkinter or PyQt.
  - ◇ Display a 9x9 grid representing the Sudoku board.
  - ◇ Differentiate between pre-filled cells and empty cells where the user can input numbers.

- **Puzzle Generation:**
  - ◇ Implement functionality to generate valid Sudoku puzzles.
  - ◇ Offer puzzles of varying difficulty levels (easy, medium, hard).
  - ◇ Ensure that each generated puzzle has a unique solution.

- **User Interaction:**
  - ◇ Allow users to select a difficulty level before starting a new game.
  - ◇ Enable users to click on an empty cell and input numbers (1-9) using the keyboard.
  - ◇ Provide the ability to erase or modify entries in the cells.
  - ◇ Implement undo and redo functionalities for user moves.

- **Game Mechanics:**
  - ◇ Enforce Sudoku rules: no duplicate numbers in any row, column, or 3x3 subgrid.
  - ◇ Provide immediate feedback if the user inputs an invalid number (optional for increased difficulty).
  - ◇ Allow users to toggle notes or pencil marks for possible numbers in a cell.

- **Hint System:**
  - ◇ Implement a hint feature that suggests a valid number for a selected empty cell.
  - ◇ Limit the number of hints available per game to maintain challenge.
  - ◇ Optionally, provide explanations with hints to help users improve their skills.

- **Solution Verification:**
  - ◇ Include a function to check if the current board is solved correctly.
  - ◇ Automatically notify the user upon successful completion of the puzzle.
  - ◇ Allow users to manually trigger a validation of their current progress.

Task Id: 25
Task: Personal Expense Tracker
Prompt: Create a Streamlit budget management tool that allows users to input their expenses and income, visualizing their spending and savings.
Requirements:

- **User Interface:**
  - ◇ Provide input fields for adding income and expense entries.
  - ◇ Include fields for amount, category, date, and description.
- **Data Management:**
  - ◇ Store the financial data in a structured format (e.g., DataFrame).
  - ◇ Allow users to edit or delete existing entries.
- **Visualization:**
  - ◇ Display a summary of total income, expenses, and savings.
  - ◇ Use charts to visualize expenses by category (e.g., pie chart, bar graph).
  - ◇ Show income and expense trends over time.
- **Budgeting Feature (Optional):**
  - ◇ Allow users to set budget limits for different categories.
  - ◇ Alert users when they are approaching or exceeding their budgets.
- **Filtering and Sorting:**
  - ◇ Enable users to filter entries by date range or category.
  - ◇ Allow sorting of entries based on different fields.
- **Data Persistence:**
  - ◇ Save the data so that it persists between sessions.
  - ◇ Provide an option to export data to a file (e.g., CSV).

Task Id: 26
Task: Vocabulary Flashcard App
Prompt: Create a Python Streamlit app to help users learn new vocabulary words using flashcards. Users can add new vocabulary and save this vocabulary.
Requirements:

- **Word Database:**
  - ◇ Maintain a list of words with definitions.
  - ◇ Allow users to add, edit, or remove words.
- **Flashcard Functionality:**
  - ◇ Present words one at a time in random order.
  - ◇ Allow users to reveal the definition after attempting recall.
- **Progress Tracking:**
  - ◇ Keep track of words the user knows and those needing practice.
  - ◇ Implement spaced repetition for effective learning.
- **User Interface:**
  - ◇ Design a simple GUI for displaying flashcards.
  - ◇ Include buttons for actions like "Know it" or "Don't know it".
- **Categories (Optional):**
  - ◇ Organize words into categories or difficulty levels.
  - ◇ Allow users to focus on specific categories.
- **Data Persistence:**
  - ◇ Save user progress between sessions.
  - ◇ Store data in a file or database.
- **Error Handling:**
  - ◇ Validate inputs when adding words.
  - ◇ Handle empty lists or other edge cases gracefully.

Task Id: 27
Task: Markdown Previewer
Prompt: Create a Streamlit markdown previewer app that lets users input markdown text, which is rendered live in a preview window.
Requirements:

- **User Interface:**
  - ◇ Provide a text input area for users to write or paste markdown content.
  - ◇ Display the rendered markdown in a separate preview area.
- **Real-Time Rendering:**
  - ◇ Implement live updating of the preview as the user types.

- ◇ Ensure minimal latency between input and rendering.

- **Supported Markdown Features:**
  - ◇ Support headings, bold, italics, lists, links, images, code blocks, and tables.
  - ◇ Handle advanced markdown features like footnotes and task lists (Optional).

Task Id: 28
Task: BMI Calculator (Console)
Prompt: Create a Body Mass Index (BMI) calculator in Python where users input their height and weight, and the app calculates and displays their BMI.
Requirements:

- **User Interface:**
  - ◇ Provide input prompts for height and weight.
  - ◇ Allow users to choose units (metric or imperial).
- **Input Validation:**
  - ◇ Ensure that the inputs are numeric and positive values.
  - ◇ Handle incorrect inputs with appropriate error messages.
- **BMI Calculation:**
  - ◇ Implement the BMI formula for both metric and imperial units.
  - ◇ Calculate the BMI based on user input.
- **Results Display:**
  - ◇ Show the calculated BMI to the user.
  - ◇ Provide a health status interpretation (e.g., underweight, normal, overweight, obese).
- **Repeat Option:**
  - ◇ Allow users to perform multiple calculations without restarting the program.
  - ◇ Include an option to exit the application.

Task Id: 29
Task: BMI Calculator (Streamlit)
Prompt: Create a Streamlit Body Mass Index (BMI) calculator where users input their height and weight, and the app calculates and displays their BMI.
Requirements:

- **User Interface:**
  - ◇ Use Streamlit widgets for input fields (text inputs, sliders).
  - ◇ Allow users to select units (metric or imperial) via radio buttons or a dropdown.
- **Input Validation:**
  - ◇ Ensure inputs are valid numbers and within a reasonable range.
  - ◇ Display error messages next to the input fields if validation fails.
- **BMI Calculation:**
  - ◇ Implement the BMI formula accordingly based on the selected units.
  - ◇ Perform the calculation when the user submits their inputs.
- **Results Display:**
  - ◇ Show the calculated BMI prominently on the app.
  - ◇ Provide a visual indicator of BMI category (e.g., color-coded text or bars).
- **User Experience:**
  - ◇ Ensure immediate feedback as users adjust inputs.
  - ◇ Design the app to be clean and easy to navigate.
- **Error Handling:**
  - ◇ Handle any exceptions silently and guide the user to correct inputs.
  - ◇ Prevent the app from crashing due to invalid data.

Task Id: 30
Task: Text Sentiment Analysis
Prompt: Create a Streamlit text sentiment analysis app where users can input text, and it analyzes the sentiment (positive, neutral, negative) using pre-trained models like VADER or TextBlob.
Requirements:

- **User Interface:**
  - ◇ Provide a text area for users to input or paste text.
  - ◇ Include a button to submit the text for analysis.

- **Sentiment Analysis:**
  - Integrate a pre-trained sentiment analysis model (e.g., VADER, TextBlob).
  - Process the input text to compute sentiment scores.
- **Results Display:**
  - Show the sentiment score or polarity to the user.
  - Classify the sentiment as positive, neutral, or negative based on the score.