

EvoDEV: An Iterative Feature-Driven Framework for End-to-End Software Development with LLM-based Agents

Junwei Liu
jwliu24@m.fudan.edu.cn
Fudan University
Shanghai, China

Chen Xu
angrytowritecode@gmail.com
Fudan University
Shanghai, China

Chong Wang
chong.wang@ntu.edu.sg
Nanyang Technological University
Singapore, Singapore

Tong Bai
22300130109@m.fudan.edu.cn
Fudan University
Shanghai, China

Weitong Chen
chenwt25@m.fudan.edu.cn
Fudan University
Shanghai, China

Kaseng Wong
alsowereme@outlook.com
Fudan University
Shanghai, China

Yiling Lou
yilinglou@fudan.edu.cn
Fudan University
Shanghai, China

Xin Peng
pengxin@fudan.edu.cn
Fudan University
Shanghai, China

Abstract

Recent advances in large language model agents offer the promise of automating end-to-end software development from natural language requirements. However, existing approaches largely adopt linear, waterfall-style pipelines, which oversimplify the iterative nature of real-world development and struggle with complex, large-scale projects. To address these limitations, we propose EvoDEV, an iterative software development framework inspired by feature-driven development. EvoDEV decomposes user requirements into a set of user-valued features and constructs a Feature Map, a directed acyclic graph that explicitly models dependencies between features. Each node in the feature map maintains multi-level information, including business logic, design, and code, which is propagated along dependencies to provide context for subsequent development iterations. We evaluate EvoDEV on challenging Android development tasks and show that it outperforms the best-performing baseline, Claude Code, by a substantial margin of 56.8%, while improving single-agent performance by 16.0%–76.6% across different base LLMs, highlighting the importance of dependency modeling, context propagation, and workflow-aware agent design for complex software projects. Our work summarizes practical insights for designing iterative, LLM-driven development frameworks and informs future training of base LLMs to better support iterative software development.

CCS Concepts

• Software and its engineering;

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Conference acronym 'XX, Woodstock, NY

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/2018/06
<https://doi.org/XXXXXXX.XXXXXXX>

Keywords

Large Language Model, End-to-End Software Development, Feature-Driven Development

ACM Reference Format:

Junwei Liu, Chen Xu, Chong Wang, Tong Bai, Weitong Chen, Kaseng Wong, Yiling Lou, and Xin Peng. 2018. EvoDEV: An Iterative Feature-Driven Framework for End-to-End Software Development with LLM-based Agents. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 14 pages. <https://doi.org/XXXXXXX.XXXXXXX>

1 Introduction

Software development has long been regarded as a complex, resource-intensive task that relies heavily on collaboration among domain experts [11, 18, 41], such as software architects, programmers, and testers. Recent advances in large language model (LLM) agents have opened up new opportunities for automatic *end-to-end software development* tasks, which aim to produce executable software directly from natural language requirements [29]. By simulating real-world development processes, multi-agent workflows can translate user requirements into executable applications.

However, most prior approaches follow a linear, waterfall-style pipeline in which requirements analysis, design, implementation, and testing are executed sequentially [7, 23, 28, 35, 42]. This workflow oversimplifies the dynamic and iterative nature of real-world software development and fails to handle the development of complex software. Some research explores the possibility of designing an agile workflow to conduct iterative software development. For example, AgileCoder [32] designs a Scrum-inspired workflow, where the software development proceeds in iterative sprints. However, it primarily relies on the underlying model to identify extension points between different iterations at the code level, failing to explicitly capture and model the complex dependencies between different requirements and provide the necessary contexts accordingly, which results in its limited capability in handling complex, large-scale software projects.

Moreover, existing works still focus on relatively simple development scenarios, such as small-scale Python applications or basic web pages, which fall short of the complexity encountered in real-world software development [29]. For instance, MetaGPT [23] has been applied to develop Python applications with fewer than 250 lines of code. In contrast, more complex development scenarios, such as Android development, pose additional challenges. Implementing features on Android often requires handling intricate application lifecycles, coordinating asynchronous tasks, managing diverse dependency configurations, and integrating with platform-specific APIs. However, the effectiveness of LLM-based agents in such more complex development scenarios remains largely unexplored.

To address these challenges, we propose EvoDev, an iterative software development framework inspired by the classic feature-driven development (FDD) methodology [20]. Specifically, we first decompose the original user requirements into a list of user-valued features and then construct a directed acyclic graph (DAG) called **Feature Map** to explicitly model the dependencies between these features. Each node in the feature map stores multi-level information about a feature, including business logic, design, and code implementation. Through the dependency relationships, this information flows to subsequent features, providing the necessary context to support smooth iterative development.

To evaluate the effectiveness and efficiency of EvoDev, we conduct experiments on more challenging Android development tasks. Experimental results demonstrate that EvoDev outperforms all existing LLM-agent baselines, and surpasses the best-performing baseline, Claude Code, by 56.8%. In addition, EvoDev brings consistent improvements to the single agent baselines by a relative improvement ranging from 16.0% to 76.6% across different base LLMs. We also summarize practical implications based on our experimental results, which might be helpful for the design of end-to-end software development frameworks and the training of base LLMs in the future.

In summary, this work makes the following contributions:

- We propose and implement the first FDD-inspired iterative software development framework, EvoDev, which constructs a global feature map to store and propagate contexts of different abstract layers through dependencies among iterations.
- We manually construct an Android software development dataset, *APPDev*, and conduct a comprehensive set of experiments to evaluate the effectiveness and efficiency of EvoDev, with the cost of approximately 500 person-hours and 1,500 USD.
- We summarize key implications of our experimental findings for future practice and research, including the demands that iterative development places on model capabilities, the misalignment of model-intrinsic behaviors with workflow guidance, and the realities of iterative development cost.

2 Background and Related Work

2.1 LLM-based agents for End-to-end Software Development

LLM-based agents have emerged as a promising solution to diverse software engineering tasks. Through enhanced capabilities in perception, planning, memory, and action, LLM-based agents can work within a code repository, read natural language requirements, plan the development process, execute code modifications via external tools, and continuously record operational trajectories in memory. This enables the agents to function as virtual programmers and complete diverse software engineering tasks, including requirements engineering [3, 39], code generation [14, 17, 27], software testing [2, 10], and debugging [22, 26, 37]. Building on this progress, recent research has explored their capabilities in more complex end-to-end tasks, such as end-to-end software development.

End-to-end software development focuses on generating target software based on user requirements from scratch. To cover diverse activities in the software development process, most existing studies emulate the real-world development process and design multi-agent workflows. Among these workflows, the linear workflow, which is inspired by the classic Waterfall process model [34], is most frequently used. It decomposes the entire development process into sequential stages such as requirements analysis, design, coding, and testing, and assigns one or more agents to each stage. Representative work includes MetaGPT [23] and ChatDev [35]. More recent work has attempted to optimize the linear workflow by incorporating reflection mechanisms to verify the alignment of intermediate outputs and make adjustments [24, 28]. In addition, some general-purpose software engineering agents like GPT-Engineer [9] and Claude Code [7], also adopt the linear workflow to ensure versatility. For example, Claude Code always outputs a linear to-do list first and completes the task step by step. However, this linear workflow oversimplifies the dynamic and iterative nature of real-world software development and faces challenges in complex, large-scale software projects. In addition, it relies heavily on the capability of base LLMs to produce a comprehensive output covering all aspects of the project.

Another research direction is to integrate an agile workflow with LLM-based agents, which is still at a primary stage. For example, AgileCoder [32] designs a Scrum-inspired iterative workflow with multiple sprints, with each sprint containing planning, development, testing, and review. However, it can only develop basic Web-based applications, with an average number of sprints being just 1.6, which is far from supporting truly iterative and incremental development. Therefore, to align with real-world software development and adapt to complex software, it is important to implement a practical iterative development framework.

2.2 Feature-Driven Development

Feature-driven development (FDD) is an agile software development methodology [20]. As shown in Figure 2, the FDD workflow starts with the construction of an overall model, which aims to analyze the business relationship of the target software and construct a coarse-grained design to keep consistency. After that, it is required

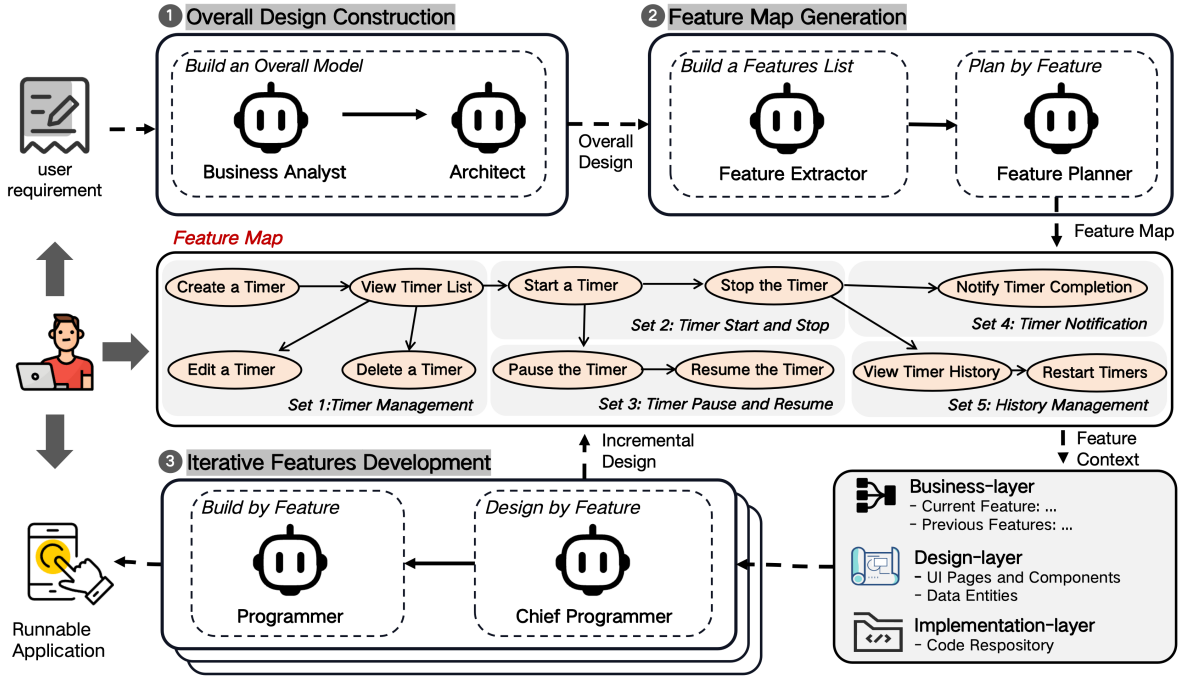


Figure 1: Overview of the FDD-inspired EvoDev framework

to extract a list of *features*, which are defined as user-valued functionalities that can be implemented within two weeks. The feature can also be integrated into feature sets, which contain a list of functionally cohesive features that can be treated as a whole. The next step is to plan by feature, where team members carefully consider the dependencies and priorities of different features and determine the development order. Finally, in each iteration, the development team makes a fine-grained design of the current feature and builds it.

Despite a classic methodology for human teams, FDD is well-suited for adaptation to a multi-agent development workflow since the input and output of all activities can be formulated in pure text format to be processed by LLM-based agents. In addition, FDD presents a promising paradigm for LLM-based agents to tackle the complexities of end-to-end software development. First, all of its activities are grounded in the initial overall design, which helps maintain consistency and alignment for LLM-based agents throughout the iterative development cycles. Second, it breaks down complex requirements into small, manageable, and verifiable features, reducing the implementation difficulty of coding agents in each iteration. Third, it includes a dedicated activity to model feature dependencies and determine development order (*i.e.*, plan by feature), which helps maintain a smooth and reliable development process.

These dependencies can further help agents to retrieve necessary contexts from preceding features, which often include the interface for implementing the current feature. Finally, it balances the global overall design with the fine-grained design within each iteration, alleviating the burden and difficulty of producing a detailed upfront design. Therefore, inspired by the strengths and adaptation, we design a multi-agent framework to simulate the FDD workflow.

3 FDD-Inspired Iterative Development Framework

In this section, we present EvoDev, an FDD-inspired iterative software development framework. As illustrated in Figure 1, EvoDev abstracts the original FDD process into three stages: **Overall Design Construction**, **Feature Map Generation**, and **Iterative Features Development**. Each stage corresponds to one or two steps in the standard FDD workflow, as indicated by the dashed boxes.

3.1 Overall Design Construction

The first stage of EvoDev is to construct an overall design of the target application based on the user requirements, which corresponds to the “*Build an Overall Model*” step in FDD. However, the user-provided requirements are often informal, with information scattered and loosely connected. To address this, we add a **Business Analyst** agent to analyze the user-provided requirements and generate a structured requirement document. As illustrated in Figure 3, the requirement document consists of two parts: (i) a concise description of the target application and (ii) a comprehensive list of all identified business workflows. Through this step, we reorganize the user-provided informal input into a structured document, facilitating subsequent data processing.

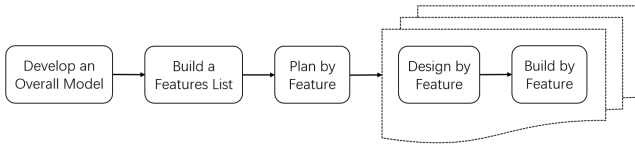


Figure 2: The basic activities of Feature Driven Development

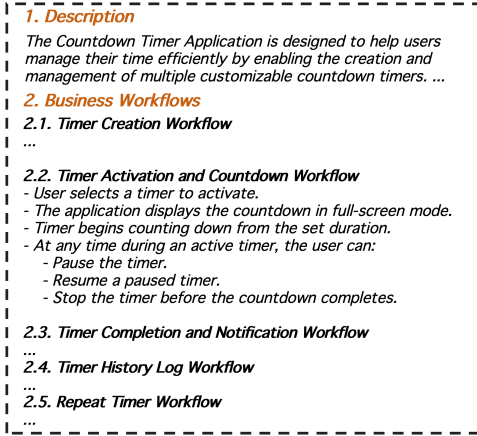


Figure 3: The requirement document for the *countdown timer APP*

Next, we introduce an **Architect** agent to construct an overall design of the target application. In FDD, the overall model refers to the domain object model. However, our preliminary experiments indicate that LLMs struggle to generate a usable domain object model, which is also reported in prior work [12, 13, 19]. Therefore, we opt to leverage LLMs for data and user interface (UI) modeling instead. A key challenge in iterative software development is that it is difficult to capture all design details in the initial stage. To address this, we instruct the Architect agent to generate an initial overall design of the target application. As illustrated in Figure 4, the overall design includes two parts: (i) a coarse-grained UI design with only the essential pages and the top-tier components on each page, and (ii) the necessary data entities. This overall design serves two primary purposes. First, it acts as the blueprint for subsequent development, preventing agents from misinterpreting the relationships between data models and UI components and arbitrarily altering them in different iterations. Second, it establishes a shared communication vocabulary among subsequent agents, which helps maintain consistency during the iterative development process.

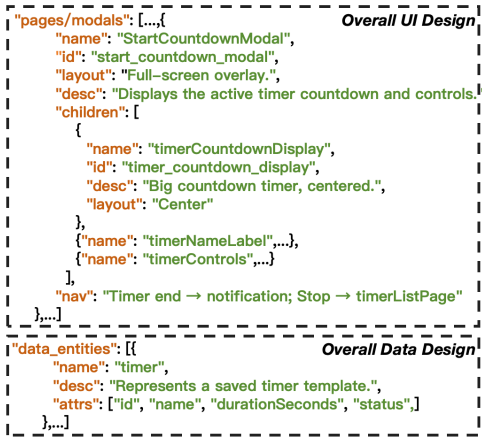


Figure 4: The overall design (including both UI and data) for the *countdown timer APP*

3.2 Feature Map Generation

Based on the constructed overall design, EVODev continues to generate a global feature dependency graph, **Feature Map**, to guide the iterative development, corresponding to the “*Build a Features List*” and “*Plan by Feature*” steps in the FDD workflow.

First, We employ a **Feature Extractor** agent to generate a detailed feature list. This agent decomposes the original requirements and overall design into a set of features, with each feature representing a small, client-valued function. To clarify the functional scope of each feature, we design a *Feature Specification Schema* to describe the functional scope of a feature, which includes the following fields:

- *Business workflow*: Describes the business workflow associated with this feature, which represents the final delivery target.
- *Business rules*: Highlights business rules that can help capture the details of the current feature, such as the default values for user data and validation rules.
- *UI flow*: Details how the user interacts with the UI components defined in the overall design.
- *Data flow*: Details how the data entities in the overall design are processed within this feature.
- *Contained data models and UI components*: Enumerates the names/IDs of all data models and UI components involved in the feature, which are used to extract the corresponding descriptions from the overall design.

To ensure consistency, all descriptions reference the IDs of UI components and data models from the overall design. Figure 5 shows the feature list of the *countdown timer* application and the description of the “*Start a Timer*” feature, with bold red text indicating the corresponding IDs.

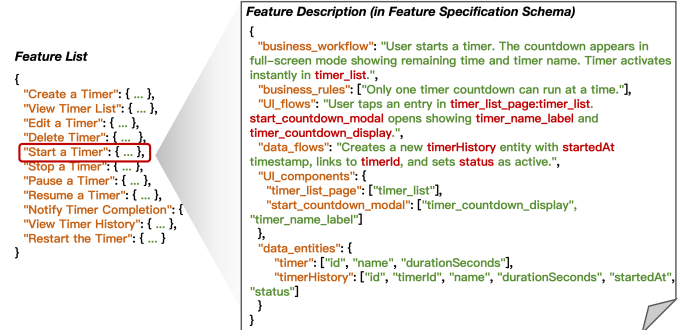


Figure 5: The feature list for the *countdown timer APP*. The **bold red** highlights indicate the UI and data designs referenced in the UI flow and data flow descriptions.

Afterwards, we assign a **Feature Planner** agent to organize the development of the extracted features. First, the Feature Planner analyzes dependencies among extracted features from both business and technical perspectives, as illustrated in the central section of Figure 1. Second, it adopts a bottom-up approach to group these features into cohesive feature sets and determines their dependencies, by following two principles: (i) features within a set should exhibit high cohesion, allowing them to be implemented within a single iteration; and (ii) feature sets must preserve the original feature-level dependencies. For example, if Feature-B depends on

Feature-A, then Feature-A must either reside in the same set as Feature-B or in a preceding set. One advantage of this approach is that the size of each feature set can be tailored to the coding capacity of different LLMs, which enhances the overall adaptability and versatility of the framework. Finally, based on the dependency relationships among these feature sets, our framework generates a directed acyclic graph (DAG) over the extracted feature sets, which is referred to as a **Feature Map**. As illustrated in Figure 6, each feature set in the feature map includes three different layers of context:

- **Business-layer Context:** this context includes the constituent features of this set and its interfaces with subsequent feature sets, specifying the functional scope of this set.
- **Design-layer Context:** this context extracts all UI components and data entities involved in constituent features from the global overall design.
- **Implementation-layer Context:** this context includes the development status, code diffs, and the modified files, and will be updated after development. It reflects the complete record of code changes involved in achieving the current business objective.

In contrast to the widely adopted linear planning strategies that merely produce a to-do list, our feature map not only captures the complex dependencies among requirements using a graph structure, but also encapsulates critical context at the business, design, and implementation layers. This enables agents to track repository changes across both temporal (feature dependencies) and spatial (design and code changes made for each feature) dimensions, thereby reducing the effort required for the agents to understand the complex code repository and improving development accuracy.

3.3 Iterative Features Development

The final stage of our framework is to iteratively develop the target application. Following the FDD methodology, where each iteration consists of “*Design by Feature*” and “*Build by Feature*” steps, we first perform a detailed design of the current feature set, and subsequently implement it based on the design specifications.

We assign a **Chief Programmer** agent to make a fine-grained design of the current feature set, which takes the business-layer and design-layer context of both the current set and its preceding sets as input and completes three tasks. First, it adopts a bottom-up method to analyze and integrate all features within the current feature set and output a feature-set-level description following the *Feature Description Schema* introduced in Section 3.2. Second, it generates incremental design descriptions for the involved components (e.g., “*Add a rounded confirm button with a green background at the bottom of the original modal.*”), which are then incorporated back into the overall design and available to subsequent iterations. This ensures that the overall design provides global guidance while allowing the Chief Programmer agent to make flexible, iteration-specific design decisions. Finally, the Chief Programmer further decomposes the fine-grained design into a development plan consisting of all tasks to complete in this iteration. In summary, the Chief Programmer reorganizes the relationships among all features in the current feature set, extends the relevant coarse-grained overall design, and produces a detailed development plan for the current iteration.

We then design a **Programmer** agent to take over responsibility for the concrete implementation. The Programmer takes the fine-grained design and development plan produced by the Chief Programmer, along with the implementation-layer context (e.g., the changes in code repository) as input, and is equipped with tools for reading, writing, and editing code files. To further improve efficiency, we design a memory mechanism to optimize the agent’s trajectory management. In particular, we observe that when an agent invokes tools, its response includes a `tool_calls` field that encompasses all the tool names and parameters. This field stores fragmented code modifications, which quickly bloat the context, introduce redundant file versions, and increase reasoning complexity. To address this, we maintain a unique memory called `file_contents` to cache the latest version of all files read, created, or edited by the agent. When the agent invokes tools, the `tool_calls` will be executed and removed, leaving merely the natural language response (e.g., “*Let’s modify [FILE_PATH] to implement the [FUNCTION]*”) in the dialog history. We then insert an extra message to report the tool execution result and notify that `file_contents` has been updated. This ensures trajectory completeness while keeping unique, always up-to-date file versions in the dialogue history, thereby maintaining a clear context with fewer tokens.

Equipped with the memory, the Programmer implements the current feature sets through two phases: coding and debugging. During the coding stage, the Programmer autonomously invokes tools to modify code in the repository. Upon completion, it outputs a special token (“`TIME_TO_END`”) to trigger the debugging phase. During debugging, the developed application is automatically built, and the error messages are returned as feedback. The Programmer reflects the implementation and applies a single-turn fix, after which the build is automatically re-run. This loop continues until all errors are resolved.

Finally, EvoDev performs a git commit, updates the development status, code diffs, and modified files of the current feature set, and deliver a runnable version of the application. The iterative development process continues until all feature sets in the feature map have been implemented and the final application is delivered.

4 Experimental Setup

In this section, we describe the setup of a comprehensive set of experiments to demonstrate the effectiveness and efficiency of our proposed EvoDev framework in iterative software development.

4.1 Research Question

We formulate the following research questions to guide our experiments:

- **RQ1-Effectiveness:** How effective is our FDD-inspired framework, EvoDev, compared with other LLM-agent-based approaches for software development?
- **RQ2-Generalizability:** Does EvoDev demonstrate consistent effectiveness in different settings?
 - **RQ2.a-Model Generalizability:** Does EvoDev demonstrate consistent effectiveness in software development across different base LLMs?

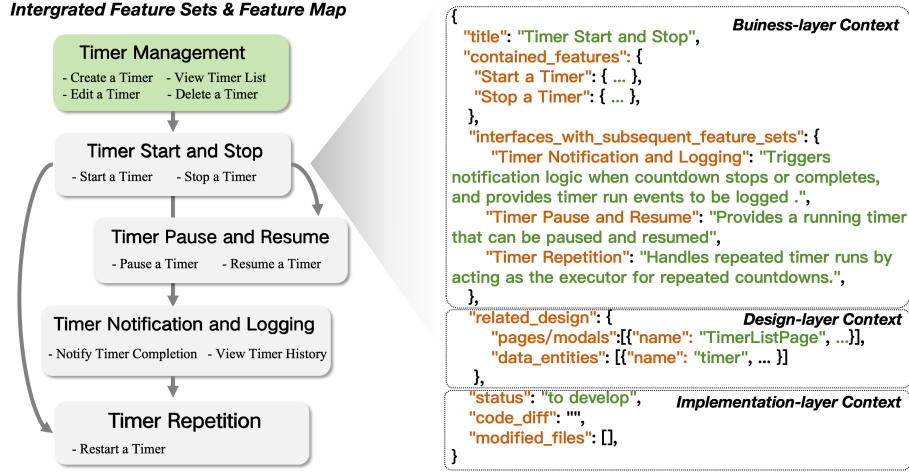


Figure 6: The feature map for the countdown timer APP, with each node containing contexts of the business, design, and implementation layers.

- **RQ2.b-Task Generalizability:** Does EvoDEV demonstrate consistent effectiveness when applied to tasks of different difficulties?
- **RQ3-Ablation Study:** How do different stages impact the performance of EvoDEV?
- **RQ4-Efficiency:** How efficient is EvoDEV, as an iterative approach, compared with other single-pass approaches for software development?

4.2 Implementation Details

Software and Language Selection. Although EvoDEV is designed to be language-agnostic, we focus on implementing a prototype for Android development. We make this decision for two reasons. First, Android powers billions of devices worldwide, making it one of the most widely adopted software platforms [1, 21]. However, it has never been evaluated in the end-to-end software development task. Second, compared to widely evaluated Python and Web development, Android development is inherently complex, involving stricter system constraints, diverse hardware interactions, and intricate application lifecycles. Moreover, the default programming language for Android development is Kotlin, which is a low-resource programming language with relatively limited corpora available for training [40]. These factors together make Android a more challenging testbed to evaluate the effectiveness, robustness, and generality of LLM-based agents in software development.

Workflow Implementation. Following the FDD-inspired framework described in Section 3, we implement EvoDEV for Android development on top of *LangGraph* [25], which is the widely used in prior work [4, 31, 38]. The workflow requires two inputs: (i) a natural language requirements description; and (ii) the path to a scaffold project generated by the official Android Studio IDE, serving as the execution environment. Except for the requirement document generated by the Business Analyst and the development plan generated by the Chief Programmer, all other intermediate outputs are stored in a structured JSON output, which is convenient for management and context retrieval. To balance effectiveness and efficiency, we limit the number of extracted feature sets to no more than four.

For tool integration, we implement tools to read, create, and edit files, and bind them to the underlying LLMs through *LangGraph*. In particular, we follow the practice of Claude Code [7] and adopt a search-substitute strategy to apply code diffs on existing files, where the agent is instructed to output both the original and revised code blocks, which are then automatically substituted into the corresponding files. The memory optimization for the programmer agent can also be implemented in *LangGraph* by integrating it into the states field of the workflow.

4.3 Evaluation Benchmark

4.3.1 Dataset. Existing works [23, 24, 35] evaluate their approaches on Python or Web development datasets, which are not suitable for Android development evaluation. Therefore, we manually construct an Android application development dataset, *APPDev*, through a three-stage pipeline:

Stage I: Application Selection. After assessing the manual efforts required for evaluation, we first fixed the size of the dataset to 15 Android applications. Then, we examined the application categories in *Google Play Store* and *Apple App Store*, and distilled five representative application categories: Utility, Lifestyle, Shopping, Education, and Entertainment. The first two authors then independently collected candidate applications from each category in both stores. Following prior work [23, 32, 35], we retained only lightweight applications whose core functionalities can be implemented within the client side without relying on backend applications or online services. The authors then discussed and finalized the three representative applications for each category.

Stage II: Requirement Specification. In the second stage, all authors conducted a collaborative brainstorming session to analyze the functional requirements of the 15 selected applications, identify the core functionalities to retain, and draft an initial version of the requirement descriptions. After that, the first two authors independently drafted the core acceptance checklists for each application and then reconciled their drafts through discussion. When consensus could not be reached, a third author was introduced as

an arbitrator. This iterative process continued until all three authors agreed on the final acceptance checklists. Simultaneously, the requirement descriptions were refined in accordance with updates to the checklist, ensuring that all functional requirements were clearly and accurately conveyed.

Stage III: Industrial Validation. To further ensure dataset quality, we invited professional practitioners from industry to review the dataset. They verified that these apps are representative and the descriptions were complete, consistent, and aligned with real-world development practices.

Table 1 summarizes the constructed dataset. For each application, we provide a detailed requirement description along with a list of functional requirements for acceptance testing. The average number of functional requirements per application is 13.5, with a maximum of 26 and a minimum of 8. Furthermore, we categorize the applications into three difficulty levels based on the number of functional requirements: *Elementary* (0–9 requirements), *Intermediate* (10–19 requirements), and *Advanced* (20 or more requirements). This categorization allows us to evaluate the LLM-based agents' performance across applications of varying complexity.

4.3.2 Baselines. We include baselines for our evaluation according to the following criteria: 1) the tool should be based on an LLM-agent architecture; 2) it should support Android application development, excluding tools that only support Python or Web development, such as ChatDev [35] and Lovable [30]; 3) it should support incremental development on an existing code repository (*i.e.*, the initial Android scaffold project); and 4) the selection should include both open-source and closed-source tools. Based on these criteria, we select three representative software development LLM-based agents as our baselines, as follows:

- **MetaGPT [23]:** An open-source multi-agent framework designed for autonomous software development. It designs a waterfall-style development workflow, where user requirements are processed through five stages (*i.e.*, analysis, design, planning, coding, and review) before producing the final application.
- **GPT-Engineer [9]:** An open-source coding agent that supports incremental modification of existing repositories based on the requirement descriptions provided in prompt files.
- **Claude Code [7]:** A state-of-the-art commercial agentic coding tool released by Anthropic, which is exclusively powered by Claude-4-Sonnet. It allows users to describe desired features in natural language and can automatically generate a to-do list, write code, and verify correctness.

Among the baselines, MetaGPT and GPT-Engineer have been frequently compared in prior work [24, 32, 35]. However, to the best of our knowledge, it is the first time that Claude Code is used as a baseline for end-to-end software development.

4.3.3 Base LLMs. We evaluate our approach using several state-of-the-art LLMs, including three proprietary models: **GPT-4.1** (gpt-4.1-20250414) [33], **Claude-3.5-Sonnet** (claude-3-5-sonnet-20241022) [6], and **Claude-4-Sonnet** (claude-sonnet-4-20250514) [8], as well as an open-source model, **Qwen3-Coder** (Qwen3-Coder-480B-A35B-Instruct) [5]. For the proprietary models, we access them via the official APIs, while for the open-source model, we use the API service provided by Alibaba Cloud [15].

4.3.4 Evaluation Procedure. In this section, we describe the procedure for the development and evaluation of the applications using different LLM-based agents.

Application Development. During the development stage, we follow the official documentation to utilize each LLM-based agent. After the user requirement and the scaffold project are configured, the agents autonomously carry out the development process. Due to limited resources, we set uniform time limits for each task based on its difficulty: 30 minutes for elementary applications, 40 minutes for intermediate applications, and 50 minutes for advanced applications. Development exceeding these limits will be automatically terminated. Finally, all generated applications are packaged and archived, and any logs produced by the agents (if available) are recorded for subsequent analysis.

Application Evaluation. A key challenge of software development tasks is that the generated applications are unpredictable, and it remains unresolved to conduct a rigorous automatic evaluation [29]. Therefore, we follow the common practice of previous studies [23, 32] and perform a manual evaluation, which is described as follows.

- *Questionnaire Design.* To collect objective feedback, we first design a Likert-scale questionnaire [36] for the application assessment, which comprises both functional and non-functional assessment questions. For the *functional* part, all requirements of an application, which have been included in the dataset, are listed, with each requirement rated on a 4-point scale: '1' indicates the function is absent, '2' indicates the function is implemented but largely incorrect, '3' indicates the function is implemented and mostly correct, and '4' indicates the function is fully implemented and correct. For the *non-functional* assessment, we follow Creswell's guidelines [16] for designing qualitative research questions and assign the first four authors to independently identify non-functional aspects that users might experience during evaluation. Through several iterative discussions, similar aspects were grouped and refined until consensus was reached. The final non-functional assessment comprises four dimensions: *visual design*, *usability*, *stability*, and *overall satisfaction*, each also rated on a 4-point scale (1 = very poor, 2 = poor, 3 = good, 4 = very good).

- *Testing Environment Construction.* To control uncertainty factors and reduce individual bias, we carefully construct a consistent testing environment for manual evaluation. First, we assigned anonymous identifiers to the applications developed by different LLM-based agents, ensuring that participants were unaware of which tool produced each application. After that, the applications were packaged as APK files and installed on the same Android testing device. Finally, we examined the prerequisites for testing each application (*e.g.*, the Music Player and PDF Reader require local music and PDF files, respectively) and prepared the required resources on the test device in advance.

- *Manual Evaluation.* We then invited four participants to individually perform a manual evaluation of all applications in sequence, *i.e.*, the assessment for the same applications should be completed before proceeding to the next. After all participants completed the assessment, we calculated the functional and non-functional scores for each application by averaging all participants' scores.

Table 1: Statistics of the Constructed APPDev Dataset. “# Req.” represents the number of functional requirements.

APP Name	Difficulty	# Req.	APP Summary	Type
Color Picker	Elementary	8	An application that allows users to import images, extract colors, and save their favorites.	Utility
QRCode Tool	Elementary	8	Allows users to scan or generate QR codes and automatically open web pages in the browser.	Utility
Expense Tracker	Elementary	8	An application for logging daily expenses, setting budgets, and generating visual reports.	Shopping
Shopping List	Elementary	8	Allows users to create shopping lists to record products they want to buy and track the changes.	Shopping
Countdown Timer	Elementary	9	An application helps users create timers and execute them to manage personal time.	Education
Decision Wheel	Elementary	9	An application that allows users to create weighted decision wheels and spin for outcomes.	Utility
PDF Reader	Elementary	9	Allows users to import PDFs from local storage or URLs, organize files into groups, and read PDFs.	Education
Music Player	Intermediate	12	This application supports importing local music, creating custom playlists, and controlling playback.	Entertainment
Trip Itinerary	Intermediate	13	The app is for creating travel itineraries, adding time-bound events, and sending event notifications.	Lifestyle
Drink Order	Intermediate	15	Allow users to browse categorized drinks, customize orders, and complete payments using a token system.	Shopping
Diary Journal	Intermediate	16	This application supports writing diaries with rich text formatting and setting daily reminders.	Lifestyle
Fitness Record	Intermediate	16	An application that lets users create workout plans, track during workouts, and view statistics.	Lifestyle
Simulated Planting	Advanced	20	A planting game allows users to plant various crops, manage soil fertility, and sell fruits to earn coins.	Entertainment
Simple Duolingo	Advanced	25	This application supports language learning, completing lessons, and tracking streaks.	Education
Shoot 'em up	Advanced	26	A classic shooting game where users control a character to defeat enemies, and upgrade abilities.	Entertainment

Overall, the entire development and evaluation process takes huge manual effort (approximately 500 person-hours) and cost (approximately 1,500 US dollars).

4.3.5 Metrics. We design the following metrics to evaluate the effectiveness and efficiency of different LLM-based agents in Android development, including functional, non-functional, and efficiency metrics.

Functional Metrics. The functional quality of the developed applications is measured by the following metrics:

- *Build Success Rate*: The proportion of developed applications that can be successfully built.
- *Function Completeness*: the average functional score across all developed applications, based on the 4-point scale defined in Section 4.3.4.

Non-functional Metrics. The non-functional experience of the developed applications is evaluated from four perspectives: *Visual Design*, *Usability*, *Stability*, and *Overall Satisfaction*. For each dimension, we report the average score collected from the Likert-scale questionnaires across all applications.

Efficiency Metrics. We report the average *Monetary* and *Time* Cost in development to measure the efficiency of different LLM-based agents. In addition, inspired by prior work [23], we design a simple relative metric, which is referred to as *Productivity*, to quantify the ability of an agent to convert its cost into functional

completeness, defined as:

$$\text{Productivity} = \frac{\text{Function Completeness} - 1}{\text{Cost}} \quad (1)$$

The subtraction of 1 shifts the minimum value of Function Completeness from 1 to 0, ensuring that an agent fails to develop any runnable application yields a Productivity of 0. Higher Productivity indicates that a unit of cost produces a greater improvement in Function Completeness. We report both the *Monetary Productivity* and *Time Productivity* in our experimental results.

5 Results and Analyses

5.1 RQ1: Effectiveness

Table 2 reports the evaluation results of EvoDev and other baselines with Claude-4-Sonnet. Both open-sourced LLM-based agents (*i.e.*, MetaGPT and GPT-Engineer) consistently fail to generate applications that can be successfully built, which highlights the non-trivial complexity of Android development and validates the challenging nature of our constructed APPDev dataset. Claude Code achieves a high build success rate of 73.3%. However, its Function Completeness remains low at 2.27. In practice, this means that while the generated code is syntactically valid, most functional requirements are merely superficially satisfied and largely incorrect. In contrast, EvoDev outperforms all baselines in both functional and non-functional metrics. It achieves a perfect build success rate of 100% and the highest Function Completeness of 3.56. Additionally, all other non-functional metrics are also superior compared to the

Table 2: Evaluation results of the generated applications for different LLM-based agents with Claude-4-Sonnet.

LLM-based Agent	Difficulty	Functional Metrics		Non-Functional Metrics			
		Build Success Rate %	Function Completeness	Visual Design	Usability	Reliability	Overall Satisfaction
MetaGPT	All	0.0	1.00	1.00	1.00	1.00	1.00
GPT-Engineer	All	0.0	1.00	1.00	1.00	1.00	1.00
Claude Code	Elementary	71.4	2.23	2.39	2.29	2.18	2.07
	Intermediate	60.0	1.95	2.05	1.95	1.90	1.80
	Advanced	100.0	2.92	2.92	2.92	3.00	3.00
	All	73.3	2.27	2.38	2.30	2.25	2.17
EvoDEV	Elementary	100.0	3.74	3.57	3.46	3.32	3.32
	Intermediate	100.0	3.64	3.90	3.75	3.65	3.65
	Advanced	100.0	3.03	3.33	3.08	3.00	3.00
	All	100.0	3.56	3.63	3.48	3.37	3.37

baselines. These results demonstrate the comprehensive improvement of EvoDEV in producing applications that are both reliable and of high quality in terms of functionality and user experience.

We further analyze the performance of EvoDEV and other baselines across tasks of varying difficulty levels. Compared to other baselines, EvoDEV consistently achieves higher performance across all metrics, with only moderate decreases as task difficulty increases. This demonstrates that EvoDEV scales robustly with task complexity and maintains reliable code generation even under more challenging scenarios. In contrast, Claude Code exhibits noticeable fluctuations. As the difficulty increases, its build success rate decreases from 71.4% to 60.0%, but then unexpectedly rises to 100.0%. A similar irregular pattern is observed in other metrics. By conducting further analysis, we identified bad cases in Claude Code’s outputs. For instance, for the *Shoot'em Up* game application from the Advanced task sets, although we invoked Claude Code within the terminal of an Android project, it still generated a Web-based application. Therefore, while the project could technically be built successfully, the generated application failed to meet any task requirement, thereby undermining the validity of its success. This example suggests that Claude Code might be task-specific overfitting. For instance, when developing game applications, it tends to generate a Web application by default and ignore the given repository structure. Such behavior might stem from biases in the underlying Claude-4-Sonnet model, which might be excessively exposed to certain patterns during training. However, Claude Code lacks effective mechanisms to regulate or adapt these tendencies, leading to outputs that deviate from the intended task requirements.

Answer to RQ1: EvoDEV demonstrates superior effectiveness compared to all existing LLM-agent-based approaches for software development. It consistently outperforms baselines in both functional and non-functional metrics, achieving a perfect build success rate and the highest Function Completeness score (3.56 with a 56.8% improvement compared to Claude Code), while maintaining robustness across tasks of varying difficulty. These results demonstrate that the FDD-inspired framework enables EvoDEV to generate not only syntactically correct but also functionally reliable and high-quality Android applications.

5.2 RQ2: Generalizability

To demonstrate the generalizability of EvoDEV, we conduct experiments to compare its performance with the single-agent setting, where only the Programmer agent is applied to develop the entire application.

5.2.1 RQ2.a: Model Generalizability. Based on the results in Table 3, EvoDEV consistently improves both functional and non-functional metrics across different LLMs, with the magnitude of improvement varying by model.

For the open-source Qwen3-Coder, the single-agent setting fails to produce runnable applications, suggesting that the model might lack the fundamental coding capability required for Android development. Consequently, although integrating our framework brings improvements, its Function Completeness remains low (1.18), which indicates that most applications it develops are unusable in practice. The Claude series demonstrates the strongest single-agent performance, with Claude-3.5-Sonnet reaching an average Function Completeness of 2.16 and Claude-4-Sonnet achieving a 100% build success rate with an average Function Completeness of 3.07. Notably, the single-agent result of Claude-4-Sonnet even surpasses Claude Code in Table 2. We hypothesize that this discrepancy arises from the mismatch between Claude Code’s linear, sequential development strategy and the underlying model’s capabilities, which ultimately limit its effectiveness. In contrast, when combined with EvoDEV, both Claude-3.5-Sonnet and Claude-4-Sonnet show consistent gains across all metrics, with absolute improvements of 0.60 (27.8% relative improvement) and 0.49 (16.0% relative improvement) in Function Completeness. Finally, GPT-4.1 demonstrates the most striking improvement when integrating with EvoDEV, with Function Completeness rising from 1.84 to 3.25 (**76.6% relative improvement**). We attribute this significant improvement to two factors. First, GPT-4.1 already achieves a high build success rate (93.3%) in the single-agent setting, reflecting its ability to generate syntactically valid Android applications. However, its Function Completeness of 1.84 reveals that it struggles to understand user requirements and model complex interdependencies among features. Our framework addresses this limitation by leveraging the feature map to enhance the model’s awareness of different project layers (business, design, implementation) and improve functional correctness. Second, GPT-4.1 benefits from its stronger general

Table 3: Evaluation Results of the generated applications for EvoDev with different LLMs.

LLM	Approach	Functional Metrics		Non-Functional Metrics			
		Build Success Rate %	Function Completeness	Visual Design	Usability	Reliability	Overall Satisfaction
Qwen3-Coder	Single Agent	0.0	1.00	1.00	1.00	1.00	1.00
	EvoDev	46.7 (+46.7)	1.18 (+0.18)	1.27 (+0.27)	1.22 (+0.22)	1.17 (+0.17)	1.17 (+0.13)
Claude-3.5-Sonnet	Single Agent	60.0	2.16	2.10	2.20	2.15	2.02
	EvoDev	93.3 (+33.3)	2.76 (+0.60)	2.62 (+0.52)	2.57 (+0.37)	2.52 (+0.37)	2.42 (+0.40)
Claude-4-Sonnet	Single Agent	93.3	3.07	3.25	3.05	2.88	2.93
	EvoDev	100.0 (+6.7)	3.56 (+0.49)	3.63 (+0.38)	3.48 (+0.43)	3.37 (+0.49)	3.37 (+0.44)
GPT-4.1	Single Agent	93.3	1.84	1.80	1.75	1.75	1.70
	EvoDev	100.0 (+6.7)	3.25 (+1.41)	2.93 (+1.13)	3.12 (+1.37)	3.22 (+1.47)	2.88 (+1.18)

instruction-following capability, which allows it to more effectively adhere to contextual guidance across iterative development. In comparison, the Claude series often performs extensive rounds of code review and repair after completing a development cycle. While this behavior increases single-round accuracy, it sometimes prematurely implements functionality scheduled for later iterations, thereby disrupting the planned workflow.

Answer to RQ2.a: EvoDev demonstrates performance gains across diverse base LLMs, consistently improving both functional and non-functional metrics, highlighting its generalization capability. Moreover, experiment results reveal that balancing the model’s programming and instruction-following capabilities is crucial for iterative development, as evidenced by the greatest improvements (76.6% relative improvement) of GPT-4.1 among all base LLMs.

5.2.2 RQ2.b: Task Generalizability. As shown in Table 4, EvoDev consistently outperforms the single-programmer agent across tasks of varying difficulty, demonstrating strong generalizability. The largest improvement is observed on Intermediate tasks for all models, followed by Elementary and Advanced tasks. For Elementary tasks, the baseline performance is already high, so the improvement is relatively small; however, after optimization, the Function Complete score reaches near-optimal levels (e.g., 3.74/4 for Claude-4-Sonnet). The Intermediate tasks appear to lie at the capability boundary of single agents. For example, Claude-3.5-Sonnet fails on six tasks in the single-agent setting, five of which are among the most difficult tasks in our dataset, suggesting that these tasks require reasoning and decomposition capabilities beyond what a single agent can provide. At this difficulty level, EvoDev effectively leverages the feature map to guide task decomposition and manage feature dependencies, enabling the agent to get necessary contexts from the business, design, and implementation layers, thereby achieving remarkable gains. For Advanced tasks, the inherent complexity might limit the improvement, and further case analysis also reveals that the modest overall improvement may be partly explained by task-specific effects. In particular, the *Shoot'em Up* application, a classic game application, appears to be overfitted by the base LLMs since all single-agent baselines achieve unusually high scores on this task (e.g., Claude-3.5-Sonnet reaches 3.76 compared to an average of 1 on other Advanced apps, GPT-4.1 reaches 3.01 compared to 1.4, and Claude-4 achieves 3.30 compared to 2.82). This

overfitting effect elevates the baseline performance on Advanced tasks. However, EvoDev still delivers consistent improvements by providing structured decomposition and dependency-aware planning, and if we exclude the *Shoot'em Up* task as an outlier, the absolute improvements of Function Completeness on Advanced tasks become more evident: Claude-3.5-Sonnet improves from 1.00 to 2.03 (+1.03), GPT-4.1 from 1.48 to 1.80 (+0.32), and Claude-4-Sonnet from 2.79 to 3.59 (+0.80). These results demonstrate that EvoDev consistently enhances performance even at the highest difficulty level.

Answer to RQ2.b: EvoDev demonstrates robust and consistent improvements across all difficulty levels, especially for complex development tasks, where single agents often reach their capability boundary and fail to capture the complex dependencies among user requirements. These results highlight the adaptability and effectiveness of EvoDev across diverse task settings.

5.3 RQ3: Ablation Study

Table 5 demonstrates that both the “Overall Design Construction” and the “Feature Map Generation” stages contribute to the final performance improvements. Despite Claude-4-Sonnet already achieving the strongest single-agent performance (i.e., 93.3% Build Success Rate and 3.07 Function Completeness), the incorporation of these stages yields consistent gains. Specifically, introducing the overall design, which establishes a coherent blueprint of the app’s structure and guides subsequent development, results in an absolute improvement of 0.22 (7.2% relative improvement) in Function Completeness. Building on this, the integration of feature maps to guide iterative development further provides an absolute improvement of 0.27 (8.2% relative improvement) on Function Completeness. This improvement is also reflected in user experience, with all non-functional metrics demonstrating similar gains. These results underscore the effectiveness of both overall design and feature map-guided iterative development in enhancing our framework.

Answer to RQ3: Both the overall design and feature map-guided iterative development contribute substantially to EvoDev’s overall performance, yielding gains of 0.22 (with a relative improvement of 7.2%) and 0.27 (a further relative improvement

Table 4: Evaluation Results of the generated applications for EvoDev across tasks of varying difficulty.

LLM	Approach	Difficulty	Functional Metrics		Non-Functional Metrics			
			Build Success Rate %	Function Completeness	Visual Design	Usability	Reliability	Overall Satisfaction
Claude-3.5-Sonnet	Single Agent	Elementary	85.7	2.78	2.57	2.71	2.86	2.57
		Intermediate	40.0	1.42	1.75	1.70	1.45	1.40
		Advanced	33.3	1.92	1.58	1.83	1.67	1.75
		All	60.0	2.16	2.10	2.20	2.15	2.02
	EvoDev	Elementary	100.0 (+14.3)	3.22 (+0.44)	2.93 (+0.36)	3.00 (+0.29)	2.96 (+0.10)	2.79 (+0.22)
		Intermediate	80.0 (+40.0)	2.50 (+1.08)	2.45 (+0.70)	2.25 (+0.55)	2.25 (+0.80)	2.15 (+0.75)
		Advanced	100.0 (+66.7)	2.10 (+0.18)	2.17 (+0.59)	2.08 (+0.25)	1.92 (+0.25)	2.00 (+0.25)
		All	93.3 (+33.3)	2.76 (+0.60)	2.62 (+0.52)	2.57 (+0.37)	2.52 (+0.37)	2.42 (+0.40)
Claude-4-Sonnet	Single Agent	Elementary	100.0	3.35	3.57	3.28	3.18	3.21
		Intermediate	80.0	2.75	2.85	2.80	2.60	2.70
		Advanced	100.0	2.96	3.17	2.92	2.67	2.67
		All	93.3	3.07	3.25	3.05	2.88	2.93
	EvoDev	Elementary	100.0 (+0.0)	3.74 (+0.39)	3.57 (+0.0)	3.46 (+0.18)	3.32 (+0.14)	3.32 (+0.11)
		Intermediate	100.0 (+20.0)	3.64 (+0.89)	3.90 (+1.05)	3.75 (+0.95)	3.65 (+1.05)	3.65 (+0.95)
		Advanced	100.0 (+0.0)	3.03 (+0.07)	3.33 (+0.16)	3.08 (+0.16)	3.00 (+0.33)	3.00 (+0.33)
		All	100.0 (+6.7)	3.56 (+0.49)	3.63 (+0.38)	3.48 (+0.43)	3.37 (+0.49)	3.37 (+0.44)
GPT-4.1	Single Agent	Elementary	100.0	2.03	2.18	2.0	1.96	1.93
		Intermediate	80.0	1.48	1.30	1.35	1.40	1.35
		Advanced	100.0	2.00	1.75	1.83	1.83	1.75
		All	93.3	1.84	1.80	1.75	1.75	1.70
	EvoDev	Elementary	100.0 (+0.0)	3.58 (+1.55)	3.07 (+0.89)	3.36 (+1.36)	3.61 (+1.65)	3.11 (+1.18)
		Intermediate	100.0 (+20.0)	3.43 (+1.95)	3.40 (+2.10)	3.40 (+2.05)	3.25 (+1.85)	3.30 (+1.95)
		Advanced	100.0 (+0.0)	2.17 (+0.17)	1.83 (+0.08)	2.08 (+0.25)	2.25 (+0.42)	1.67 (-0.08)
		All	100.0 (+6.7)	3.25 (+1.41)	2.93 (+1.13)	3.12 (+1.37)	3.22 (+1.47)	2.88 (+1.18)

of 8.2%) in Function Completeness, respectively, on the best-performing Claude-4-Sonnet model.

5.4 RQ4: Efficiency

Table 6 presents both the monetary and time cost for different LLM-based agents. First, on both GPT-4.1 and Claude-3.5-Sonnet, our approach achieves comparable monetary and time costs to the single-agent baselines. In addition, the higher Productivity metrics indicate that although our tool introduces extra overhead, it yields higher function completeness under the same cost. For instance, on GPT-4.1, we incur an extra 0.39 dollars and around 4 minutes per application, yet achieve a 76.6% improvement in Function Completeness, demonstrating the effectiveness of our method.

We attribute this improvement to three factors. First, the single agent retains the entire development context in its dialogue history, which grows with project size and increases token costs. In contrast, our approach stores hierarchical information in the feature map and provides only the relevant context for each iteration, reducing overhead caused by cluttered contexts. Feature decomposition also simplifies the difficulty of implementation, decreasing the time spent on repeated debugging. Second, as described in Section 3.3, we optimize the single-agent trajectory to prevent dialogue histories from being polluted with fragmented code modifications and redundant file versions, further reducing token costs. Third, we limit the feature set to at most four based on preliminary experimental results, balancing effectiveness and overhead.

The only exception occurs in experiments with Claude-4-Sonnet as the base LLM, where EvoDev’s average cost (\$4.63) matches Claude Code (\$4.38) but is roughly double that of the single agent (\$1.56), and the average development time (around 23 minutes) is

also higher. We assume that this deviation stems from Claude-4-Sonnet’s default development behavior. As we mentioned in RQ2.a, Claude-4-Sonnet tends to perform multiple rounds of code review and repair in development, which improves the single-pass generation accuracy but increases time and token usage. To validate this hypothesis, we randomly select two applications (*i.e.*, *Shopping List* and *Countdown Timer*) and manually check the number of development rounds of both GPT-4.1 and Claude-4-Sonnet. The result shows that Claude-4-Sonnet performs 12 rounds per feature, while GPT-4.1 only performs 5 rounds.

We further analyze the open-source baselines, MetaGPT and GPT-Engineer, which both fail to produce any runnable applications. Among all the approaches, MetaGPT incurs the highest overhead (\$9.61 and around 24 minutes per app). This might be due to its excessive design; for example, even for the elementary *QRCode Tool* app, it generated 47 files and over 7,000 lines of code. In contrast, GPT-Engineer performs minimal design and code modification, with an average code lines of less than 350. This demonstrates that even with state-of-the-art LLMs, the design of the development workflow critically affects outcomes, with overly complex or overly simple designs both leading to failure. Our approach strikes a balance between workflow design and code implementation, consistently improving performance across different scenarios.

Answer to RQ4: On GPT-4.1 and Claude-3.5-Sonnet, EvoDev achieves comparable efficiency performance to single-agent baselines while delivering higher Function Completeness per unit cost. However, the underlying agent behaviors of Claude-4-Sonnet make it tend to perform fine-grained step-by-step development, which requires more modification rounds per feature. Iterative development further amplifies this issue and

Table 5: Ablation study of EvoDev with Claude-4-Sonnet. “Iterative/Single-pass” indicates that the agent performs iterative development when the feature map is available, and single-pass generation otherwise.

Overall Design Construction	Feature Map Generation	Iterative/Single-pass Development	Functional Metrics		Non-Functional Metrics			
			Build Success Rate %	Function Completeness	Visual Design	Usability	Reliability	Overall Satisfaction
✗	✗	✓	93.3	3.07	3.25	3.05	2.88	2.93
✓	✗	✓	100.0 (+6.7)	3.29 (+0.22)	3.43 (+0.18)	3.33 (+0.28)	3.25 (+0.37)	3.22 (+0.29)
✓	✓	✓	100.0 (+0.0)	3.56 (+0.27)	3.63 (+0.20)	3.48 (+0.15)	3.37 (+0.12)	3.37 (+0.15)

incurs more cost, highlighting the need to balance the model-intrinsic behaviors with the guidance of the external workflow.

Table 6: Average cost of different LLM-based agents in developing applications in APPDev

LLM	Approach	Monetary Metrics		Time Metrics	
		Cost (\$)	Productivity	Cost (min)	Productivity
GPT-4.1	Single Agent	0.63	1.43	10.98	0.08
	EvoDev	1.02	2.19	14.50	0.15
Claude-3.5-Sonnet	Single Agent	2.07	0.58	14.52	0.08
	EvoDev	2.88	0.61	18.23	0.10
Claude-4-Sonnet	MetaGPT	9.61	0.00	23.84	0.00
	GPT-Engineer	0.09	0.00	1.28	0.00
	Claude Code	4.38	0.27	11.81	0.10
	Single Agent	1.56	1.37	9.18	0.23
	EvoDev	4.63	0.57	22.65	0.12

6 Discussion

6.1 Implications

We summarize the following insightful implications based on the findings in our experiments.

Iterative Development Rewards Balanced Capabilities. The coding capability is regarded as the key metric to measure the performance of LLMs in software engineering tasks. However, our results reveal that while the Claude series (*i.e.*, Claude-3.5-Sonnet and Claude-4-Sonnet) demonstrates superior single-agent coding capabilities, GPT-4.1 benefits far more from the iterative development framework due to its stronger global planning and instruction-following capabilities. Our experimental results on Qwen3-Coder also suggest that the coding capability remains a bottleneck. This highlights a tradeoff between enhancing the coding capabilities and instruction-following capabilities of LLMs.

Model-Intrinsic Behaviors May Misalign Workflow Guidance. Our experiments across different LLM-agent-based baselines reveal a potential mismatch between model-intrinsic and workflow-guided behaviors. As we described in RQ1 and RQ4, Claude-4-Sonnet insisted on a multi-round self-editing and revision behavior pattern, even when we set explicit prompting constraints (*e.g.*, avoiding self-revision and fixing errors based on the build feedback). This behavior pattern strengthens its coding success rate, but sometimes also disturbs the workflow plan and incurs more overhead. We hypothesize that this behavior might be because it is overfitting to this behavior pattern during its training to achieve higher coding performance. However, such model-intrinsic agent behaviors can conflict with those expected by the external workflows, demonstrating the misalignment of training more autonomous agents with the design of downstream agent systems.

Iterative Development Can Be Cost-Efficient. Iterative workflows intuitively appear more costly and less efficient. However, our

experiments show that when applied to GPT-4.1 and Claude-3.5-Sonnet, the additional overhead remains within acceptable bounds. Moreover, when measuring the function completeness gained by unit cost, iterative approaches can even surpass single-pass generation in efficiency. Therefore, iterative development can also be cost-efficient and might be a promising future paradigm for sustainable AI-assisted software engineering.

6.2 Threats to Validity

Our study may face several threats to validity, which we discuss as follows.

Internal Threats. The internal threats lie in our manual evaluation to assess the functional completeness and non-functional experiences of the developed applications, which might incur individual biases. To mitigate subjectiveness and mistakes, we carefully design a Likert-scale questionnaire to assist in collecting objective feedback and invite four participants to calculate the average scores.

External Threats. There are some threats that impact the generalizability of our study. First, our dataset is of a limited size due to the extensive evaluation cost. However, we try our best to design a rigorous dataset construction process and invite industrial experts to verify the dataset. These efforts mitigate the risks of bias and improve the objectivity and applicability of our dataset. Second, our implementation and evaluation of EvoDev are specific to Android development with Kotlin. As a result, the findings may not be generalizable to other types of applications and programming languages. Exploring the iterative software development process for different languages is a valuable direction for future research.

6.3 Limitations

We then discuss the limitations of our framework. First, we do not incorporate human-in-the-loop interaction in the current approach. Nevertheless, our framework provides multiple extension points for such interaction. For example, users can inspect and modify the feature map at any time, and after each iteration, users can input new requirements that can be integrated into the feature map for subsequent development. We plan to incorporate this capability in future work. Second, our method does not include testing. Our preliminary experiments show that integrating testing in Android remains challenging for LLM-based agents. It is difficult for them to generate correct test code, and when test failures occur, they struggle to determine whether the issue lies in the original implementation or the generated tests. Integrating an effective testing mechanism in iterative development is a promising research direction.

7 Conclusion

This work proposes EvoDev, a novel iterative framework for end-to-end software development. Inspired by classic feature-driven development methodology, EvoDev constructs a feature map to explicitly stores the context of different abstract layers within a project and manages complex dependencies among different requirements, which helps provide necessary contexts for each iteration. Experimental results show that EvoDev not only surpasses existing baselines but also significantly improves the performance of single agents across different base LLMs, with an acceptable overhead and higher productivity. Further analysis reveals insightful implications for balancing the coding and instruction-following capabilities of LLMs in iterative development and raises the concern about the misalignment of agent-intrinsic with framework-guided behaviors.

Acknowledgments

This work was supported by Huawei Technologies Co., Ltd.

References

- [1] Andi Fitriah Abdul Kadir, Arash Habibi Lashkari, and Mahdi Daghmehchi Firoozjaei. 2024. *Android Operating System*. Springer Nature Switzerland, Cham, 25–42. doi:10.1007/978-3-031-48865-8_2
- [2] Azat Abdullin, Pouria Derakhshanfar, and Annibale Panichella. 2025. Test Wars: A Comparative Study of SBST, Symbolic Execution, and LLM-Based Approaches to Unit Test Generation. In *2025 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 221–232.
- [3] Yasaman Abedini and Abbas Heydarnoori. 2025. Leveraging Large Language Models for Classifying App Users' Feedback. *arXiv preprint arXiv:2507.08250* (2025).
- [4] S Akilesh, Rajeev Sekar, Om Kumar CU, D Prakalya, and M Suguna. 2025. Multi-Agent hierarchical workflow for autonomous code generation with Large Language Models. In *2025 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCECS)*. IEEE, 1–5.
- [5] as Qwen series is developed by Alibaba Cloud) Alibaba Cloud (implied. 2024. Qwen3 Coder - Agentic Coding Adventure. Web page. <https://qwen3lm.com/>
- [6] Anthropic. 2024. Introducing computer use, a new Claude 3.5 Sonnet, and Claude 3.5 Haiku. Web page. <https://www.anthropic.com/news/3-5-models-and-computer-use>
- [7] Anthropic. 2025. *Claude Code*. <https://www.anthropic.com/claude-code>.
- [8] Anthropic. 2025. Claude Sonnet 4. Web page. <https://www.anthropic.com/claude/sonnet>
- [9] AntonOsika. 2025. *GPT-Engineer*. <https://github.com/AntonOsika/gpt-engineer>.
- [10] Benoit Baudry, Khashayar Etemadi, Sen Fang, Yogya Gamage, Yi Liu, Yuxin Liu, Martin Monperrus, Javier Ron, André Silva, and Deepika Tiwari. 2024. Generative AI to generate test data generators. *IEEE Software* 41, 6 (2024), 55–64.
- [11] M Bialy, V Pantelic, J Jaskolka, A Schaap, L Patcas, M Lawford, and A Wasssyng. 2017. Software engineering for model-based development by domain experts. In *Handbook of system safety and security*. Elsevier, 39–64.
- [12] Kua Chen, Yujing Yang, Boqi Chen, José Antonio Hernández López, Gunter Müssbacher, and Dániel Varró. 2023. Automated domain modeling with large language models: A comparative study. In *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 162–172.
- [13] Ru Chen, Jingwei Shen, and Xiao He. 2024. A Model Is Not Built By A Single Prompt: LLM-Based Domain Modeling With Question Decomposition. *CoRR abs/2410.09854* (2024). arXiv:2410.09854 doi:10.48550/ARXIV.2410.09854
- [14] Matteo Ciniselli, Alberto Martin-Lopez, and Gabriele Bavota. 2024. On the generalizability of deep learning-based code completion across programming language versions. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*. 99–111.
- [15] Alibaba Cloud. 2024. Tongyi Large Language Models: The First Choice for Enterprises Embracing the AI Era. Web page. <https://www.aliyun.com/product/tongyi>
- [16] John W Creswell and J David Creswell. 2017. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications.
- [17] Leuson Da Silva, Jordan Samhi, and Foutse Khomh. 2025. LLMs and Stack Overflow discussions: Reliability, impact, and challenges. *Journal of Systems and Software* (2025), 112541.
- [18] Eric Evans. 2004. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional.
- [19] Yingqiang Ge, Wenyue Hua, Kai Mei, Juntao Tan, Shuyuan Xu, Zelong Li, Yongfeng Zhang, et al. 2023. Openagi: When llm meets domain experts. *Advances in Neural Information Processing Systems* 36 (2023), 5539–5568.
- [20] Sadhna Goyal. 2007. Agile techniques for project management and software engineering. In *Major Seminar on Feature Driven Development*. 1–19.
- [21] Renda Han. 2024. Survey: The Evolution and Future of Android Software Development. *Deep Learning and Pattern Recognition* 1, 1 (2024).
- [22] Stefanus A Haryono, Ferdian Thung, David Lo, Lingxiao Jiang, Julia Lawall, Hong Jin Kang, Lucas Serrano, and Gilles Muller. 2021. Androevolve: Automated update for android deprecated-api usages. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 1–4.
- [23] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiaowu 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 A Multi-Agent Collaborative Framework. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. <https://openreview.net/forum?id=VtmBAGCN7o>
- [24] Yue Hu, Yuzhu Cai, Yaxin Du, Xinyu Zhu, Xiangrui Liu, Zijie Yu, Yuchen Hou, Shuo Tang, and Siheng Chen. 2025. Self-Evolving Multi-Agent Collaboration Networks for Software Development. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net. <https://openreview.net/forum?id=4R71pdPBZp>
- [25] LangChain Inc. 2024. LangGraph: Balance Agent Control with Agency. Web page. <https://www.langchain.com/langgraph>
- [26] Hong Yi Lin, Chunhua Liu, Haoyu Gao, Patanamom Thongtanunam, and Christoph Treude. 2025. CodeReviewQA: The Code Review Comprehension Assessment for Large Language Models. *arXiv preprint arXiv:2503.16167* (2025).
- [27] Junwei Liu, Yixuan Chen, Mingwei Liu, Xin Peng, and Yiling Lou. 2024. STALL+: Boosting LLM-based Repository-level Code Completion with Static Analysis. *CoRR abs/2406.10018* (2024). arXiv:2406.10018 doi:10.48550/ARXIV.2406.10018
- [28] Jie Liu, Guohua Wang, Ronghui Yang, Mengchen Zhao, and Yi Cai. [n. d.]. AltDev: Achieving Real-Time Alignment in Multi-Agent Software Development. ([n. d.]).
- [29] Junwei Liu, Kaixin Wang, Yixuan Chen, Xin Peng, Zhenpeng Chen, Lingming Zhang, and Yiling Lou. 2024. Large Language Model-Based Agents for Software Engineering: A Survey. *CoRR abs/2409.02977* (2024). arXiv:2409.02977 doi:10.48550/ARXIV.2409.02977
- [30] Lovable. 2025. *Lovable*. <https://lovable.dev/>.
- [31] Shiraj Mandulapalli, Emilio Hernandez, Wayne Jordan Hall, Alireza Chakeri, and Luis Jaimes. 2025. Development of Agentic Workflows with LangGraph for Software Development Life Cycle Automation. In *North American Conference on Industrial Engineering and Operations Management-Computer Science Tracks*. Springer, 45–54.
- [32] Minh Huynh Nguyen, Thang Phan Chau, Phong X. Nguyen, and Nghi D. Q. Bui. 2025. AgileCoder: Dynamic Collaborative Agents for Software Development based on Agile Methodology. In *IEEE/ACM Second International Conference on AI Foundation Models and Software Engineering, Forge@ICSE 2025, Ottawa, ON, Canada, April 27-28, 2025*. IEEE, 156–167. doi:10.1109/FORGE66646.2025.00026
- [33] OpenAI. 2025. Introducing GPT-4.1 in the API. Web page. <https://openai.com/index/gpt-4-1/>
- [34] Kai Petersen, Claes Wohlin, and Dejan Baca. 2009. The waterfall model in large-scale development. In *International Conference on Product-Focused Software Process Improvement*. Springer, 386–400.
- [35] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ChatDev: Communicative Agents for Software Development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 15174–15186. doi:10.18653/V1/2024.ACL-LONG.810
- [36] John Robinson. 2024. Likert scale. In *Encyclopedia of quality of life and well-being research*. Springer, 3917–3918.
- [37] Adriana Sejfia, Satyaki Das, Saad Shafiq, and Nenad Medvidović. 2024. Toward improved deep learning-based vulnerability detection. In *Proceedings of the 46th IEEE/ACM international conference on software engineering*. 1–12.
- [38] Purva Sharma and Jayakumar Kaliappan. 2025. Optimised Intelligent Software Company Management System using Multi-Agent Framework. *Grenze International Journal of Engineering & Technology (GIJET)* 11 (2025).
- [39] Syed Tauhid Ullah Shah, Mohammad Hussein, Ann Barcomb, and Mohammad Moshirpour. 2025. Explainability as a Compliance Requirement: What Regulated Industries Need from AI Tools for Design Artifact Generation. *arXiv e-prints* (2025), arXiv–2507.
- [40] Sergey Titov, Mikhail Evtikhiev, Anton Shapkin, Oleg Smirnov, Sergei Boytsov, Daria Karaeva, Maksim Sheptyakov, Mikhail Arkhipov, Timofey Bryksin, and Egor Bogomolov. 2024. Kotlin ML Pack: Technical Report. *CoRR abs/2405.19250*

- (2024). arXiv:2405.19250 doi:10.48550/ARXIV.2405.19250
- [41] Jim Whitehead. 2007. Collaboration in software engineering: A roadmap. In *Future of Software Engineering (FOSE'07)*. IEEE, 214–225.
- [42] Simiao Zhang, Jiaping Wang, Guoliang Dong, Jun Sun, Yueling Zhang, and Geguang Pu. 2024. Experimenting a New Programming Practice with LLMs.

CoRR abs/2401.01062 (2024). arXiv:2401.01062 doi:10.48550/ARXIV.2401.01062

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009