An Empirical Study on Low Code Programming using Traditional vs Large Language Model Support

Yongkun Liu[a], Jiachi Chen[a,*], Tingting Bi[b,c], John Grundy[d], Yanlin Wang[a], Jianxing Yu[e], Ting Chen[f], Yutian Tang[g], Zibin Zheng[a,h]

[a]*School of Software Engineering, Sun Yat-sen University, China*
[b]*The University of Melbourne, Australia*
[c]*The University of Western Australia, Australia*
[d]*Faculty of Information Technology, Monash University, Australia*
[e]*School of Artificial Intelligence, Sun Yat-sen University, China*
[f]*Shenzhen Institute for Advanced Study, University of Electronic Science and Technology of China, China*
[g]*University of Glasgow, United Kingdom*
[h]*Zhuhai Key Laboratory of Trusted Large Language Models, China*

Abstract

Low-code programming (LCP) refers to programming using models at higher levels of abstraction, resulting in less manual and more efficient programming, and reduced learning effort for amateur developers. Many LCP tools have rapidly evolved and have benefited from the concepts of visual programming languages (VPLs) and programming by demonstration (PBD). With the huge increase in interest in using large language models (LLMs) in software engineering, LLM-based LCP has begun to become increasingly important. However, the technical principles and application scenarios of traditional approaches to LCP and LLM-based LCP are significantly different. Understanding these key differences and characteristics in the application of the two approaches to LCP by users is crucial for LCP providers in improving existing and developing new LCP tools and in better assisting users in choosing the appropriate LCP technology. We conducted an empirical study of both traditional LCP and LLM-based LCP. We analyzed developers' discussions on Stack Overflow (SO) over the past three years and then explored the similarities and differences between traditional LCP and LLM-based LCP features and developer feedback. Our findings reveal that while traditional LCP and LLM-based LCP share common primary usage scenarios, they significantly differ in scope, limitations, and usage throughout the software development lifecycle, particularly during the implementation phase. We also examine how LLMs impact and integrate with LCP, discussing the latest technological developments in LLM-based LCP, such as its integration with VPLs and the application of LLM Agents in software engineering.

Keywords:
Low code programming, Large language model, Empirical Study

1. Introduction

Low-code programming (LCP) has recently become a discussed topic in software development communities [1]. It primarily aims to minimize the use of low-level textual programming languages and utilize higher levels of model abstractions that align more closely with natural human thought processes for developing software [2]. By applying LCP, professional developers can reduce time-consuming and repetitive manual coding, and thereby improve development efficiency. LCP also allows amateur developers, referring to individuals with minimal or no formal programming training, to achieve programming objectives with much lower learning costs and effort [3, 4].

Recently, Hirzel [2] categorized the latest LCP technologies into three primary technologies: visual programming languages (VPLs) [5], programming by demonstration (PBD) [6], and programming by natural language (PBNL) [7]. VPLs allows the user to create programs by manipulating program elements graphically. PBD means that the system can record tasks performed manually by the user via a keyboard and mouse, subsequently generating programs that perform replicate these actions. PBNL translates user-provided natural language text into an executable program. The main technologies currently applied by popular LCP tools in the industry were VPLs and PBD. For example, Microsoft PowerApps [8] uses VPLs to enable users to design user interfaces and workflows using drag-and-drop visualization modules, bypassing the need for manual programming. Uipath [9] employs PBD to automate processes. With its built-in logger, users can capture UI mouse movements and keyboard activities, generating precise automation scripts.

The recent emergence of large language models (LLMs) has

*Corresponding author
Email addresses: liuyk39@mail2.sysu.edu.cn (Yongkun Liu), chenjch86@mail.sysu.edu.cn (Jiachi Chen), Tingting.Bi@unimelb.edu.au (Tingting Bi), John.Grundy@monash.edu (John Grundy), wangylin36@mail.sysu.edu.cn (Yanlin Wang), yujx26@mail2.sysu.edu.cn (Jianxing Yu), brokendragon@uestc.edu.cn (Ting Chen), yutian.tang@glasgow.ac.uk (Yutian Tang), zhzibin@mail.sysu.edu.cn (Zibin Zheng)

Table 1: Terminology and Definitions

Term	Definition
Traditional LCP	Platforms that reduce coding effort through components, visual interfaces, or demonstration, covering PBNL, VPL, and PbD as subcategories.
LLM-based LCP	Low-code platforms powered by large language models, often combined with PBNL or VPL to support natural language or visual programming.
PBNL	Programming by Natural Language: creating programs via natural language instructions; a subcategory of Traditional LCP and a core mechanism in many LLM-based LCPs.
VPL	Visual Programming Language: programming through graphical interfaces (e.g., drag-and-drop, flowcharts); a common form of Traditional LCP that can also integrate with LLMs.
PbD	Programming by Demonstration: creating programs by user demonstrations that the system abstracts into logic; a subcategory of Traditional LCP.

validated the capability of PBNL-based approaches to assist software development [10]. LLMs are widely used to generate code across various programming languages using natural language and achieve great performance on various programming tasks, e.g., code generation and code completion [11, 12], which significantly enhances software development efficiency. We define this type of LCP as LLM-based LCP.

With the growing popularity of LLMs, PBNL has also gained significant attention. Developers can now engage with LLMs like ChatGPT to implement LCP by using natural language. In this paper, we refer to LCP involving prevalent VPLs and PBD as "***traditional LCP***". In contrast, LCP based on LLMs and programming by natural language is termed as "***LLM-based LCP***". To enhance clarity and ensure consistent terminology, we include a glossary (Table 1) that defines and summarizes the key concepts used throughout this paper—namely Traditional LCP, LLM-based LCP, VPL, PBD, and PBNL—and explicitly notes their areas of overlap.

Both "traditional LCP" and "LLM-based LCP" have their advantages and disadvantages. Traditional LCP is intuitive, readable, and unambiguous for users, which significantly reduces syntax errors and some simple programming errors. However, it usually has limited application scenarios. For example, PowerApps can only be used for web development. In contrast, LLM-based LCP, with its vast repository of programming knowledge encompassing scripts, algorithms, and more, offers a wide range of applications. This makes it more accessible for amateur developers to create applications. Nevertheless, the generated programs may suffer from inaccuracies due to mis-understanding user prompts, the hallucination problem of LLMs, and erroneous generated code, leading to errors.

To understand the key characteristics and limitations of traditional LCP and LLM-based LCP, we analyze the posts on Stack Overflow (SO) [13], which is the most popular and widely used question and answer (Q&A) platform for developers to ask and answer questions. We first applied 27 keywords related to traditional LCP and 8 keywords pertinent to LLM-based LCP to preliminary filter out related posts. This process produced a substantial data set of 7,367 posts on traditional LCP and 7,468 posts on LLM-based LCP over the last three years. To further refine our dataset and ensure relevance to our research objectives, we employed ChatGPT, utilizing custom-designed prompts. Finally, we performed a manual analysis on the 1,688 posts filtered (642 and 1,046 posts for traditional and LLM-based LCP, respectively) [14] to answer the following three research questions:

RQ1.What application domains do discussions on traditional LCP and LLM-based LCP focus on?

Answering this RQ enhances our understanding of the distinct characteristics of the two LCPs and provides insights into their primarily usage by current users. The analysis for this RQ was conducted using data from 470 posts related to traditional LCP and 566 posts pertaining to LLM-based LCP. Our findings reveal that LLM-based LCP covers a broader spectrum of application domains compared to traditional LCP, many of which focus on web development. Users typically employ LLM-based LCP for addressing general programming challenges, whereas traditional LCP is used more frequently to resolve API integration issues.

RQ2. To which software development tasks do traditional LCP and LLM-based LCP contribute?

There are several phases in the software development life cycle, i.e., requirement analysis & planning, design, implementation, deployment, testing, maintenance. It is essential to determine the stages in which developers are more actively involved in discussions about traditional LCP and LLM-based LCP. Such insights can reveal where each LCP type offers the most benefit. We used the constant comparison method to collect 412 instances related to traditional LCP, and 559 instances related to LLM-based LCP. Subsequently, we classified them into their respective software development life cycle phases. We found that both traditional LCP and LLM-based LCP discussions focused on the implementation phase, and the former focused more on the deployment phase than the latter.

RQ3. What are the limitations associated with traditional LCP and LLM-based LCP?

Understanding the limitations encountered by users of the Traditional LCP and LLM-based LCP respectively is critical. This knowledge is instrumental for LCP providers in shaping the development of future tools. In addressing this research question, we analyzed a total of 65 instances related to LLM-based LCP and 95 instances concerning traditional LCP. From this analysis, we identified seven limitations for traditional LCP and five for LLM-based LCP, as reported in SO data. Our comparative analysis identified the most critical limitations suggested for each LCP type. For instance, users of LLM-based LCP grapple with higher demands for technical expertise and face more pronounced reliability concerns. Such insights are invaluable for understanding the distinct challenges posed by each LCP type, guiding providers in tailoring their development strategies to better meet user needs and expectations.

In summary, our main contributions are as follows:

- We compare LLM-based LCP with traditional LCP and

identified their unique characteristics, application domains and limitations.

- We discuss how LLMs drive the development of traditional LCP tools, how the concept of VPLs drives the development of LLM-based LCP, and the impact of LLMs agent on LCP.
- We publish our dataset and results at https://zenodo.org/records/11232842 to facilitate further studies.

The rest of this paper is structured as follows. We introduce some background knowledge and concepts in Section 2. In Section 3, we elucidate the methodologies employed for investigating LCP. Section 4 unveils the results and noteworthy findings gleaned from our thorough exploration into LCP. Section 5 provides an in-depth discussion of the latest LLM-based LCP technologies and future developments. Section 6 introduces related work. The conclusion and future work are articulated in Section 7.

2. Background

2.1. Low-Code Programming

The term "low-code" first appeared in a report published by Forrester Research in 2014 [15]. The original meaning of the term was that applications could be developed quickly with minimal manual coding and minimal up-front investment in setup, training, and deployment. However, in subsequent reports and research on LCP, the definition of the term "low-code" remained in a constant state of flux and ambiguity [16, 17, 18]. To address this ambiguity, the definition proposed by Hirzel [2] is adopted in this paper, which emphasizes that LCP are technologies designed to minimize text-based programming, with the core aim of reducing reliance on manual coding, whether through VPLs, PBD, or PBNL.

Whether for professional developers or amateur developers, the purpose of LCP is to save manual programming time and reduce the learning cost of manual programming as much as possible [19]. Traditional LCP tools, such as Power Apps [8] and OutSystems [20], typically include a visual development interface for drag-and-drop development. They are often equipped with cloud services for automated deployment, but cloud deployment is a common feature rather than a defining requirement of LCP.

2.2. Large Language Models

LLMs, such as CoPilot [21] and ChatGPT [22], refer to deep learning decoder models trained on massive amounts of text, and increasingly other data like images and models, typically based on the Transformer architecture [23]. In the realm of software development, LLMs have emerged as transformative tools, showcasing remarkable capabilities across various tasks, including code repair, code generation, test case generation, documentation and code summarization [24, 25]. A key example of an AI-assisted programming tool based on LLMs is GitHub Copilot [21]. By interpreting comments and understanding the context of the code being written, GitHub Copilot is capable of suggesting relevant code snippets and even completing entire code, thereby expediting the coding process and reducing the learning cost on developers. A number of LLM agents have also been proposed specifically for software generation recently, which can generate code and software throughout the software development life cycle [26, 27]. This shows that LLMs show strong potential in the LCP field.

2.3. Prompt Design

In this paper, "prompt design" refers to crafted text descriptions used to guide LLMs (e.g., ChatGPT) in generating their output. Recent LLMs can engage in multi-turn conversations out of the box and accomplish diverse, complex tasks specified in the input text. However, research has shown that the quality of LLMs' responses and user satisfaction can be strongly influenced by prompt design [28]. The same communicative intent may elicit comprehensive, detailed, and helpful responses or responses that are unhelpful or even incorrect, depending entirely on how the prompt is designed. Therefore, designing a well-crafted prompt is crucial for effectively guiding LLMs in completing a given task.

2.4. Software Development Processes

The software development life cycle (SDLC) is used to plan and manage the process of software development [29]. It typically involves breaking down the software development work into smaller, parallel, or sequential steps or subprocesses to improve the development process. There are many models of SDLC, usually encompassing key tasks including planning, requirements engineering, design, implementation, testing, deployment, and maintainance tasks. Software project planning includes team formation, tool selection, estimation, contracting and others. Requirements engineering involves understanding and document the client's needs and planning the overall strategy for the software development process. Requirements engineering tasks involve understanding and documenting the client's needs, as well as planning the overall strategy of the software development process. Design tasks include developers deciding the architecture and design of the software, including interface, behaviour and data, providing detailed explanations of how it will meet the requirements. During implementation tasks, developers code the software product. Testing tasks include the process of verifying and confirming the software to ensure it meets the requirements and has no vulnerabilities. Deployment involves the release of the software and making it available, typically involving installation and configuration processes. Finally, there are maintenance tasks, where developers may update the software, fix any issues that arise, and possibly add new features or functionalities.

3. Methodology

Figure 1 shows an overview of the process used to derive answers to our three research questions (RQs) from Stack Over-

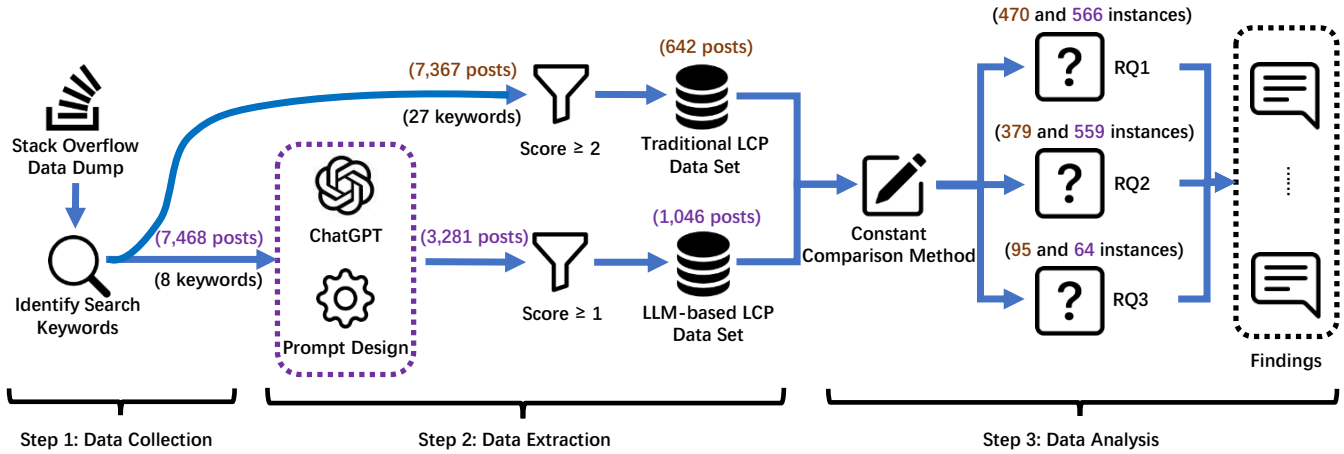

Figure 1: Overview of the investigation process, each step corresponds to a subsection in Section 3

flow (SO) posts. Step 1 involves downloading the entire historical dataset of posts from Stack Overflow. We then use a set of specifically chosen keywords to preliminarily filter LCP-related posts. In Step 2 we focus on the design of effective prompts and apply ChatGPT to further refine the data. This step ensures the removal of irrelevant data, ultimately obtaining the traditional LCP dataset with 642 posts and the LLM-based LCP dataset with 1,046 posts. In Step 3 we use the constant comparison method to manually analyze the data and obtain the answer of the three RQs.

3.1. Step 1: Data Collection

3.1.1. SO Posts Collection

We downloaded the entire SO data dump [30] of September 2023, which includes content contributed by users on the SO network. The SO data dump consists of five XML documents: Badges.xml, Comments.xml, Users.xml, Posts.xml, and Votes.xml. For our purposes, we used the Posts.xml data file, which contains comprehensive details of all posts, including the unique ID of each post, type (question or answer), title, etc. The data dump included posts spanning from July 2008 to September 2023. Notably, the groundbreaking GPT-3 [10] LLM was released in June 2020, and prior to this, there was little discussion on LLM on SO. To ensure the timeliness and consistency of the data supporting the research results, we utilized data from the last three years (June 2020 to September 2023).

3.1.2. Keywords-Based Post Filtering.

After collecting all historical posts, we conducted keyword-based filtering to select the data. For traditional LCP, we used the term "low-code" and a list of terms relevant to some popular LCP tools as keywords. For LLM-based LCP, we employed eight LLM-related keywords for further filtering. Additionally, we applied case-ambiguity matching in our keyword filtering to enhance the inclusivity of our search.

- **Keywords for traditional LCP**: To capture the discussions about traditional LCP by developers, we searched not only for the term "low-code" but also included names of top low-code tool. This approach is based on the observation that developers often mention specific tool names when discussing difficulties encountered while using LCP tools on SO. We compiled a list of top low-code tools from a previous work [31] and the top 10 platforms from G2 website (A popular website to review software) [32]. The list of top low-code tools contained 24 tool names. We deliberately excluded the term "no code" as a keyword, since it often introduces substantial noise (e.g., "no code available", "no code error") and yields very few relevant posts, as also reported by Luo et al. [63]. The final set of keywords was as follows:

```
["Zoho Creator", "App Maker", "Salesforce
    Lightning", "Quickbase", "OutSystems",
    "Mendix", "Vinyl", "Appian", "PowerApps
    ", "UiPath", "Claris Filemaker", "
    Servicenow", "Salesforce", "IAR
    Embedded Workbench", "Airtable", "Pega
    ", "Appy Pie", "Glide", "Nintex", "
    AppSheet", "AppMySite", "Softr", "Ninox
    ", "Quixy", "Low Code", "Low-Code", "
    Lowcode"]
```

Listing 1: Keywords related to traditional LCP

- **Keywords for LLM-based LCP**: To identify posts related to LLM-based low-code programming, we used "LLM" and a list of top LLM-based tools as keywords. Note that many posts retrieved in this way were not directly relevant to LCP, thus we needed further filtering, which is detailed in Section 3.2. Based on the latest works [33, 34], we selected the most popular LLM-based tools and code generation tools as keywords. The final search keywords included:

```
["LLM", "LLMs", "GPT", "ChatGPT", "Copilot
    ", "Bard", "Codex", "Claude"]
```

Listing 2: Keywords related to LLM-based LCP

This keyword-based approach to analyzing SO data from the past three years yielded a substantial number of posts: 7,367 related to traditional LCP and 7,468 pertaining to LLM.

3.2. Step 2: Data Extraction using ChatGPT.

After the above steps, we obtained the preliminary dataset related to traditional LCP and LLM-based LCP. However, the dataset may contain many posts irrelevant to LCP. For example, some posts that contain LLM-related keywords may discuss the fine-tuning process of LLMs and not usage for LCP. Given the large volume of such data, manual data filtering can be both time-consuming and error-prone. Since ChatGPT [22] has demonstrated excellent natural language comprehension and text classification capabilities [35], we employed ChatGPT to help filter data.

3.2.1. Prompt Design

We optimized our ChatGPT prompt strategy based on some existing prompt strategies [36, 37, 38]. Specifically, we first let ChatGPT analyze each post data set sentence to ascertain its relevance to LCP. Then, ChatGPT was instructed to deliver an answer (True or False) based on the analysis results, which facilitates large-scale data filtering. The prompt design details are shown below:

> ***User Prompt 1:*** You are **[ROLE]**. **[KNOWLEDGE DESCRIPTION]**. **[TASK DESCRIPTION]**. Think step by step, carefully. The input statements are as follow: **[INPUT]**.

> ***ChatGPT Response 1:*** **[ANALYSIS]**.

> ***User Prompt 2:*** **[OUTPUT FORMAT]**.

> ***ChatGPT Response 2:*** **[TRUE/FALSE]**.

where the placeholder **[ROLE]** in ***User Prompt 1*** denotes the specific role assigned to ChatGPT. In this study, we designated the role as an expert in the field of software engineering. In addition, we described the concepts and knowledge about LCP in the placeholder **[KNOWLEDGE DESCRIPTION]**, ensuring that ChatGPT has accurate prior knowledge to accomplish the task. We assigned a task to ChatGPT in the placeholder **[TASK DESCRIPTION]**, which involved analyzing sentence by sentence whether the input content was related to LCP. Finally,we entered the post to be assessed in the placeholder **[INPUT]**. ChatGPT then returned the result of a sentence-by-sentence analysis **[ANALYSIS]** in ***ChatGPT Response 1***. In the placeholder **[OUTPUT FORMAT]** of ***User Prompt 2***, we required ChatGPT to assess whether each sentence in the **[ANALYSIS]** is related to LCP. Output **TRUE** if any sentence is relevant, otherwise output **FALSE**. The full text of the prompt design and results is available in our open source repository.

3.2.2. Filter by Posts' Score

Through our data filtering using ChatGPT, we reduced the dataset of LLM-based LCP from 7,468 posts to 3,281 posts. However, this is still a large volume for human analysis. To facilitate higher quality data selection, we ranked posts based on their scores in SO [1]. A higher score typically indicates a higher quality discussion. In this paper, we applied score thresholds of score≥1 and score≥2 for the traditional LCP dataset and LLM-based LCP dataset, respectively. The rationale is twofold: (1) posts with non-positive scores were excluded to remove low-quality content; and (2) to ensure fairness, we ranked posts by score and included not only the top 500 but also all posts tied at the cutoff score. After this additional filtration, our datasets comprised 642 traditional LCP posts and 1,046 LLM-based LCP posts.

3.3. Step 3: Data Analysis

To analyze the collected data qualitatively, we employed the constant comparison method [14]. This method was originally proposed by Glaser and Strauss and has been widely applied in qualitative research. Its core idea is to continuously compare new and existing data during the analysis process, thereby gradually deriving higher-level concepts and categories. In this study, our application of the constant comparison method consisted of the following steps: (1) In the initial phase, the first author summarized the collected data, aligning it with the specific RQs. This involved assigning codes to individual posts, capturing the content's essence. (2) Subsequently, the first author aggregated these codes into higher-level concepts. These concepts were then transformed into overarching categories, fostering a more abstract representation of the information. (3) To enhance the reliability and robustness of the analysis, the second author meticulously reviewed the assigned codes and the derived categorizations. We calculated Cohen's Kappa coefficient to quantify inter-rater agreement, which reached 0.92, indicating strong consistency between coders. In cases of disagreement, the two authors discussed until consensus was reached; if consensus could not be immediately achieved, the disagreement was revisited until alignment was obtained. (4) To effectively code and categorize data, we used the qualitative data analysis tool MAXQDA [39], a tool that facilitates coding text and abstracting these codes into conceptual categorizations.

It is worth noting that despite the use of keywords and ChatGPT for filtering, some posts unrelated to LCP may still be present. Fortunately, all filtered data underwent a manual analysis process, during which we removed any irrelevant posts. For instance, in post *A76965127*, the user discussed the robustness of LLM's embedding vector representations to subtle differences, which was unrelated to LLM-based LCP. After manual review and labeling of two datasets, we identified 451 and 230 posts unrelated to LCP, respectively. Consequently, in the traditional LCP and LLM-based LCP datasets, the actual number of posts discussing LCP is 411 and 595, respectively. This means that at the ChatGPT screening stage, the accuracy of ChatGPT in identifying LCP-related posts was 56.9%. While the precision was relatively low due to our intentional prompt design

[1] The score of a post is determined by user votes in SO and is calculated as $Score = Upvotes - Downvotes$, which reflects both content quality and community recognition.

favoring higher recall, the method achieved a recall of 87.18%, which ensured that most potentially relevant posts were captured. Although this introduced some false positives, the subsequent manual verification effectively corrected them, thereby mitigating the impact of noise. As a result, the final datasets reached both a reliable quality and a sufficient scale to support further qualitative analysis.

4. Results

4.1. Answer to RQ1: What application domains do discussions on traditional LCP and LLM-based LCP focus on?

We applied the constant comparison method and we obtained 470 instances related to the traditional LCP, distilled into four application scenario categories. We also obtained 566 instances related to the LLM-based LCP, categorized into 9 distinct application scenario categories. The distribution of the discussions on traditional LCP and LLM-based LCP in various application scenarios was presented in Tables 2 and 3, respectively.

Application Domains. Table 2 shows that the discussions of the traditional LCP have only four categories. They focus on API integration, data management, web backend and web customization. In contrast, Table 3 shows that the distribution of discussions regarding LLM-based LCP are spread across 9 distinct categories. Notably, the predominant areas of focus include general issues, web frontend, web backend, and data management, which collectively account for 79.32% of the discussions. Additionally, this table also highlights the varied nature of LLM-based LCP applications in other domains, such as algorithms, gaming, and programming scripts. This analysis shows that traditional LCP is more focused on application domains related to web development, whereas LLM-based LCP exhibits a broader range of application domains.

> ***Finding 1:*** *LLM-based LCP encompasses a wider array of application domains compared to traditional LCP, which are more narrowly focused on web development.*

General Issues and API Integration. The top-ranked categories in Table 2 and Table 3 highlight the prevalent topics in traditional LCP and LLM-based LCP. General issues are a significant focus in LLM-based LCP discussions, accounting for 32.86%, whereas API integration takes the lead in traditional LCP discussions, reaching a substantial 36.60%. This sheds light on the distinct challenges within these two LCPs.

In the category of general issues, users discussed issues they may face in various application areas. For example, they addressed issues such as regular expressions, character format conversion functions, and misspelled variable names. These issues typically involve basic programming skills and syntax usage. This means that in LLM-based LCP, users often use LLMs to generate code to solve these basic general problems, which may be difficult for novice developers to solve in manual programming. In the category of API integration, users discussed various API integration issues they encountered when using traditional LCP tools. For example, in Q63364119, the user encountered an error when trying to add a custom button for calling an external API. Since the underlying code in traditional LCP tools is all encapsulated into APIs, the most common problems users face often shift from generic programming problems to API integration problems.

In summary, the main discussions on traditional LCP and LLM-based LCP are related to general usage issues. The issues faced by LLM-based LCP manifest as general programming issues, while the issues faced by traditional LCP manifest as API integration issues. Both LCPs can prevent users from getting bogged down in the details of low-level code, with LLM generates professional code and traditional LCP encapsulates expert code within API.

> ***Finding 2:*** *The main discussions about traditional LCP and LLM-based LCP revolve around general usage issues. The former predominantly involves API integration issues, whereas the latter centers on general programming challenges.*

Main Application Domains. In the analysis of traditional and LLM-based LCP LCP discussions, a notable observation is the top four categories remained consistent across both contexts. These encompassed web frontend, web backend, and data management, signifying their universal importance in the LCP application domain.

Many discussions related to traditional LCP usage involve developers seeking help on SO in areas where they encounter problems while using LCP tools. For example, developers sought assistance due to a lack of response when clicking buttons (Q62646468) or inquiring about how to filter databases in the LCP tool (Q63319755). Most discussions related to LLM-based LCP are about how to use LLMs to assist in the development of these areas. For example, using Copilot to implement the functionality of changing string colors with a button click (Q74792689), or using ChatGPT to generate code for manipulating JSON files (Q76614479). Typically, beginners seek the help of LLMs to generate solutions when they want to implement a feature but did not know where to start.

When comparing the total proportion of these categories in Table 2 versus Table 3, we found that they accounted for 63.40% in the discussions about traditional LCP and 46.46% in the discussions about LLM-based LCP. This revealed that both LLM-based LCP and traditional LCP users often leverage them to accomplish tasks related to web development and data management. This proportion is basically in line with the proportion of developers in reality. According to JetBrains' report in 2022 [40], 75% of developers were involved in web development, while developers in other fields are relatively fewer. The proportion of traditional LCP was higher than that of LLM-based LCP, due to the fact that traditional LCP tools are usually dedicated to quickly building web applications and do not involve other domains such as algorithms and games.

Table 2: Distribution of the SO discussion on traditional LCP across application areas

Application Areas	Proportion(%)	Description	Example
API integration	36.60	Discussion related to using API functionality within the LCP tools.	*Q67883356: I use the client ID (=AppID) and token in the ArcGIS JavaScript API like below:...Only implementing this gives me an error...*
Web Backend	24.26	Discussion related to building logical flows, function usage, and authentication within LCP tools.	*Q63850758: I'm currently attempting to build an Airtable-esque filter component.*
Data Management	23.40	Discussions related to database management and data manipulation.	*Q63319755: How to use LINQ on a DataTable in Uipath?...*
Web Customization	15.74	Discussion related to UI, buttons, and component customization.	*Q62646468: I am able to trigger the first button click and I am able to cause an alert with the second button click, but for some reason when I try to trigger the first button click in the click event handler of the second button, it doesn't work.*

Finding 3: *Users predominantly focus on similar application areas regardless of traditional LCP or the LLM-based LCP, i.e., web frontend, web backend and data management.*

4.2. Answer to RQ2: To which software development tasks do traditional LCP and LLM-based LCP contribute?

We identified and analyzed a total of 379 instances related to traditional LCP and a total of 559 instances associated with LLM-based LCP. We categorized these instances into the corresponding stages of the software development life cycle (requirement analysis & planning, design, implementation, testing, deployment, and maintenance), resulting in Table 5 and Table 4. Discussions related to these two LCPs spanned the entire software development life cycle tasks.

Implementation Tasks. By observing Tables 5 and 4, it can be seen that the implementation phase dominated in both types of discussions. Traditional LCP accounted for 94.99%, while LLM-based LCP accounted for 87.66%. This phase involved actual application development, where traditional LCP typically included tasks such as customizing the user interface, implementing business logic, and debugging the implemented functionality. For example, in A67962911, users discussed how to change the expression type from *int32* to *String* during the visual development process on the UiPath Studio platform, which is part of implementing business logic. Tasks completed during this phase with LLM-based LCP were more extensive. In addition to the mentioned tasks, they also encompassed algorithm debugging, script writing, hardware development, and more. For instance, in A76040047, the user utilized GPT4 for debugging and identifying errors occurring in encoding RSA keys.

The concentration of discussions on implementation tasks in Tables 5 and 4 is likely due to Stack Overflow's nature as a technical Q&A forum that actively fostered conversations around programming techniques. Developers frequented the platform to delve into issues related to the practical aspects of coding, making the implementation phase the focal point of discussions.

Finding 4: *Discussion posts on usage of both traditional LCP and LLM-based LCP are prevalent across the software development life cycle, with a particular emphasis on the implementation-related tasks.*

Deployment Tasks. A distinction was observed in deployment phase discussions between traditional LCP (1.06%) and LLM-based LCP (7.87%). This indicates that compared with traditional LCP users, LLM-based LCP users were more concerned about deployment related topics in the software development life cycle. This disparity aligns with the intrinsic differences between the two types of LCP. Traditional LCP typically integrated seamlessly with cloud platforms, streamlining deployment operations through automation. This automation spared users the need to delve into the intricate details of the deployment process, creating a more user-friendly experience. On the contrary, a feature of LLM-based LCP is in its ability to generate code that still requires a manual configuration process during the deployment phase.

Finding 5: *LLM-based LCP discussions show a heightened focus on deployment-related tasks compared to traditional LCP, reflecting the different needs and concerns of users in this phase of the software development life cycle.*

4.3. Answer to RQ3: What are the limitations of traditional LCP and LLM-based LCP?

Our analysis identified a total of 95 instances related to traditional LCP and organized them into 4 categories (Table 6). We also identified a total of 65 instances pertaining to LLM-based LCP, which were subsequently organized into 7 distinct categories (Table 7).

Reliability Concerns. In Table 7, the predominant limitation observed was the reliability concern, constituting 33.33% of the instances. Users expressed concerns about the reliability of generated code, as illustrated in the case of Q69918631. Despite GitHub Copilot generating code that adhered to user instructions, concerns about the code's reliability and potential errors persisted. Due to users lacking sufficient professional knowledge and LLMs lacking adequate reliability, users

Table 3: Distribution of the SO discussion on LLM-based LCP across application areas

Application Areas	Proportion(%)	Description	Example
General	32.86	General issues encountered during development using LCP, such as function format, regular expression, debugging and so on.	*A76930839: When asked correctly at ChatGPT it answers:I can see the issue in your code. There is a small typo in the RETURN statement for the Celsius conversion. You have misspelled the variable name temperature as temperature.*
Web Frontend	16.78	Discussion of elements related to HTML, CSS, UI interfaces, etc.	*Q74792689: ...Every time a button is clicked, it should change its color, while the other buttons should remain the same ...So magically Copilot suggested this, and it works ...*
Web Backend	15.90	Discussion of server back-end and logical flow design for web application development.	*A75778070: ...with some guidence from ChatGPT I found this which fixes exactly what I needed. ... chose the REST API route to point to the frontend and not the backend system ...*
Data Management	13.78	Discussions related to database management and data manipulation.	*Q76614479: I only wish to have one additional column with the information under 'dataSet"id' ... asked Chatgpt, it gave me this code which looks correct but just generate Traceback*
Programming script	8.82	Discussion of programming scripts that implement automation or other functions.	*Q76773098: ...I want to try out Excel Scripts. Being less familiar with JavaScript, I thought I'd try ChatGPT to do the heavy lifting...*
Algorithm	6.54	Discussion of algorithm design or implementation.	*Q76739321: I am running a simulation based on moving mass involving rayleigh beam system ... The following boundaries to be used areI have tried running this in jupyter notebook with the help of chat gpt ...*
Game	2.30	Discussion of game development.	*Q76701048: I'm new to coding and the game developing scene, so for my first project, I'm trying to make a Mario esc platformer in Godot. ... This occurred after asking GPT to help me add friction. This is what the code looks like now...*
Hardware	1.59	Discussion of hardware or embedded programming.	*Q76205204: I'm trying to send AT commands to HC05 for get other device names and rssi values.I'm using MSP-EXP430G2ET and HC05 module ... I asked Chat GPT to write this ...*
Others	1.41	Discussions that are too specific or niche to categorize.	*Q73684154: ...I used GitHub Copilot to help me write this next part, as I know I need to use some math to get a frequency out of this...*

were unable to assess the correctness and quality of the code and were concerned about the possibility of causing significant losses.

Similarly, Table 6 shows that users of traditional LCP also exhibited concerns about reliability, at a slightly lower rate of 15.78%. Users' doubts about the reliability of traditional LCP were usually related to the platform's APIs or components. If these components had quality or security issues, the stability and reliability of the developed applications would be affected. For example, in A63836762, the user mentioned a kind of unreliability: since Salesforce's CometD protocol implementation did not support acknowledgment (ack), therewas a risk that subscribers will not receive messages. Users needed to design their solutions to locate and replay events that had not been committed to the target database.

The data related to reliability concerns in Tables 7 and 6 indicated that both LLM-based LCP and traditional LCP, especially the former, frequently encountered reliability issues. The concerns for LLM-based LCP stemmed from the inherent uncertainties and limitations of LLMs, while those for traditional LCP were related to the quality of components and potential issues with external APIs.

Finding 6: *Users of both traditional and LLM-based LCP express concerns over reliability, with a higher degree of apprehension observed in LLM-based LCP due to the uncertainties associated with LLMs.*

Need for Professional Knowledge. LCPs aim to simplify programming tasks for professional developers and empower amateur developers with limited coding experience. The intention is to facilitate the implementation of various functions without delving into the intricacies of traditional coding. However, despite the advancements in LCP, the findings presented in Tables 6 and 7 shed light on the fact that users still require professional programming knowledge and skills, particularly when dealing with more complex functionalities.

Table 6 and Table 7 show that the percentage of instances expressing the "Need for professional knowledge" amounted to 15.94% and 34.74%, respectively. Notably, this statistic represented the highest proportion in Table 6, indicating a significant reliance on programming knowledge in traditional LCP.

Table 4: Distribution of the discussion on traditional LCP through the software development life cycle.

Life cycle	Proportion(%)	Description	Example
Implementation	94.99	Take place actual coding and development of the software.	*A67962911: While the expression type for a defaults to int32, not String, it can be easily changed by opening the Properties tab in UiPath Studio and clicking the drop-down box associated with the TypeArgument attribute.*
Testing	2.11	Verify and validate the software to ensure it meets the requirements and is free of bugs.	*Q67406513: I'm currently using cypress to test salesforce, and I'm running into a certain circumstance where I don't know the Party record ID that will create it within the opportunity...*
Deployment	1.06	Release the software and made available for use, often involving installation and configuration processes.	*Q73917687: This code collects data from Airtable, and creates pdf file to download.It works perfectly on my localhost.But when I deploy it to Heroku server, it throws an error...*
Design	0.79	Outline the software's architecture and design, and detail how it will meet the requirements.	*Q75826438: the webserver contacts a platform, which is used for hosting (low-code platforms such as OutSystems or CAMUNDA are closest to what I mean here), in order to run a particular Service A. This communication is managed by Event Management so that the Webserver never gets directly in contact with the service.Now, I want to represent my application as a single white-boxed component and create a context view...*
Maintain	0.79	Update the software, fix any issues that arise, and potentially add new features or functionality.	*Q68092704: ...Google has changed something in recent weeks relating to 'caching of data in Google Sheets'. I have tried reaching out to Google Workplace support to raise this issue but they don't seem to care, and they don't offer any support for GAS...*
Project Planning & Requirements Engineering	0.26	Understand and document the client's needs and planning the overall strategy for the software development process.	*Q76176735: I want a password protected repository that compiles as a website for documentation, orientation, and teaching for a team of student interns... Does anyone have a suggestion about an open source solution to this?...*

As exemplified in A76286719, the user pointed out that utilizing ChatGPT for constructing a simple Alloy model was mostly accurate. However, the user still emphasized the necessity of having a fundamental understanding of Alloy model creation, reinforcing the idea that even with LLM-based LCP, users still needed to grasp foundational concepts. Similarly, in A63730131, the user believed that the LCP tool UiPath fundamentally provided the possibility of a no-code experience. Nevertheless, when confronted with intricate issues that demand the leveraging of code or the creation of custom activities, users still found themselves in need of a solid grasp of C# knowledge to write the necessary code.

In summary, for users aspiring to incorporate more sophisticated features, the understanding of professional development skills remains crucial. LLM-based LCP users needed to understand basic programming concepts and knowledge to guide LLM in generating practical code and to validate the reliability of the generated code. Meanwhile, traditional LCP users had to possess programming skills to break free from the limitations of basic components and APIs, enabling them to implement more advanced and complex functionalities.

Finding 7: *In order to implement advanced functions, traditional and LLM-based LCP users need professional programming knowledge, with traditional LCP users needing more.*

Version Problem. Another significant issue identified in Table 6 and Table 7 was the "version problem", accounting for 17.39% and 13.68%, respectively.

Traditional LCP encountered many version-related problems, albeit for different reasons. In traditional LCP tools, users typically relied on the APIs or components provided by the platform and its suppliers to implement custom functionalities. However, when a component underwent an upgrade and conflicts with some common components, users might find that their custom functions no longer operate as intended. An insightful instance was highlighted in A76189225, users in Salesforce mentioned that conflicted between "many organizations are still using outdated pages and components" and "common component upgrades" resulted in errors without warnings.

In LLM-based LCP, version issues arose due to the static nature of LLM training data, which may not align with current library updates and programming environments. As a result, the generated code could be inconsistent with the libraries and environments used by the user. Prompting issues might produce less than accurate LCP code. Additionally, the hallucination of LLM might lead to incorrect versions of functions. These issues resulted in version-related limitations for users of LLM-based LCP. An illustrative example of this limitation could be found in Q76189225, where the user attempted to implement certain functions with the assistance of code generated by ChatGPT. However, the generated code turned out to be outdated.

Overall, version issues faced by LLM-based LCP tended to

Table 5: Distribution of the discussion on LLM-based LCP through the software development life cycle.

Life cycle	Proportion(%)	Description	Example
Implementation	87.66	Take place actual coding and development of the software.	*A76132989: Nevermind, ChatGPT came to the rescue. The solution turned out to be using a combination of a inside a (this allows flutter to optimize the individual slivers) and the package for adding the headers (using , the aformentioned doesn't support slivers as far as I'm aware).*
Deployment	7.87	Release the software and made available for use, often involving installation and configuration processes.	*Q75503643: I am upgrading my app from Rails 6.1 to Rails 7. While testing it on development env. , it works fine. But I encouter an issue while uploading it on heroku. Heroku raise this error while precompiling assets :I've been trying many solutions from stackoverflow and ChatGPT...*
Design	1.79	Outline the software's architecture and design, and detail how it will meet the requirements.	*Q76154923: I need to model a computation task and some sub-tasks depend on it... ChatGPT suggests that I could define this kind of structure as fixed point, so that I can make use of cata to fold it...*
Testing	1.43	Verify and validate the software to ensure it meets the requirements and is free of bugs.	*Q75776530: I'm trying to write tests for an Arduino program, using VSCode with Platformio...So ChatGPT gave me a hint of using a mock digital writebut then told me to which would work fine in Python, but not in C++.*
Maintain	0.89	Update the software, fix any issues that arise, and potentially add new features or functionality.	*Q74959869: If I merge custom to company_b (and it just so happens that I don't get a merge conflict) I lose the changes made in company_b. How can I deal with this? According to ChatGPT I could dobut , who knows. I tried just in case and the changes I made in company_b, specifically deleting a part of the code, are missing and the code I deleted is back.*
Requirement Analysis & Planning	0.36	Understand and document the client's needs and planning the overall strategy for the software development process.	*Q74962245: As far as I know, If I want to create a new database, normally fistly I have to create and switch on a server ... I'm using ChatGPT to help me realized about any variables I don't know (Although it sometimes gives wrong information, so I try to contrast it)...*

be more critical, often leading to project compilation failures. In contrast, traditional LCP's version issues primarily arose during the maintenance phase, requiring providers to promptly maintain components and resolve version conflicts.

Finding 8: *Both traditional LCP and LLM-based LCP users face version-related issues, with the former due to conflicts during component upgrades, and the latter stemming from the outdated data and hallucinate of LLM.*

API Function. In Table 6, it was noteworthy that API function related discussions constituted a substantial portion, ranking the second position at 18.95%. This underscored a common challenge faced by traditional LCP that heavily relied on predefined APIs, where users encountered functional constraints due to the inherent limitations of these interfaces. For example, in A70005251, the user encountered API performance limitations, which might posed a bottleneck for developers seeking to implement high-level customization of functionality within their applications.

Compared to traditional LCP, LLM-based LCP significantly deviated from these limitations. LLM-based LCP had strong learning capabilities and the ability to autonomously generate code. Its powerful learning ability enabled it to adapt to constantly changing user requirements, no longer constrained by fixed API interfaces, thus handling more complex scenarios. In contrast, traditional LCP relied on static APIs and could not proactively adapt to changing user needs, resulting in a noticeable disadvantage in flexibility. Furthermore, when implementing a customized functionality, LLM-based LCP could provide solutions and automatically generated code. In contrast, traditional LCP often required developers to manually write a large amount of code when dealing with functional customization, increasing the development workload.

Finding 9: *LLM-based LCP appears to surpass traditional LCP's API limitations, enhancing flexibility and adaptability.*

Data Migration. Data migration also played a high role in the discussion of traditional LCP, reaching 16.84%. Data migration referred to the fact that users might want to migrate data to another LCP platform because of API limitations, functional limitations, etc. However, due to domain-specific languages (such as the specific query languages for each platform), migrating data or functionality from one platform to another became quite complex.

For example, in Q74056413, the user attempted to migrate data and reports from the sales platform to another platform. But Salesforce had SOQL (salesforce object query language),

Table 6: Limitations of traditional LCP Discussed on SO

Limitation	Proportion(%)	Description	Example
Need for professional knowledge	34.74	Users need professional programming knowledge to use LCP tools well.	*A63730131: if you go further and having a complex issue that can only be solved with invoking code or creating custom activities, you really need to code.*
API function	18.95	Users may meet error when using API function.	*A70005251: Unfortunately, airtable has a fixed 5 requests per second limit for all pricing levels. There's no way to increase this limit.*
Data migration	16.84	Platform closure and domain-specific languages make data migration difficult for users.	*Q74056413: I am relocating our Salesforce reports from Salesforce to another tool. Salesforce has SOQL and the new system has SQL so I think the simplest way to migrate is to modify the SOQL statements to SQL, rather than recreate each report using the new tool's UI.Can I do this?*
Reliability concern	15.79	The components or APIs of the LCP tool is unreliable.	*A63836762: Salesforce's implementation of CometD doesn't support ACKs. Even if it did, you'd still have ...but the frequency/loss of risk might be lower.In your case you have to engineer a solution that amounts to finding and replaying events that were not committed to your target database.*
Version problem	13.68	The LCP tool may cause errors due to library or API version issues.	*A67410760: many orgs continue using obsolete pages and components. However, public Apex classes and public Visualforce components are deleted as part of the upgrade process. If you delete pages and components without performing this two-stage procedure, Salesforce can't warn you when later deletions of public classes and components break your subscribers obsolete pages and components Emphasis mine.*

Table 7: Limitations of LLM-based LCP Discussed on SO

Limitation	Proportion(%)	Description	Example
Reliability concern	33.33	The code generated by the LCP tool is unreliable.	*Q69918631: The GitHub Copilot has suggested I should use the following algorithm:So, I've tried with some testing dates and, the result is always correct. However, I'd like to know whether if this is a good way to make date comparisons or it may lead to a wrong response at any moment?*
Version Problem	17.39	The LCP tool may cause errors due to library or API version issues.	*Q76189225: ...I even tried using chatGPT but the code generated is confusing and somehow outdated so its not working*
Need for professional knowledge	15.94	Users need professional programming knowledge to use LCP tools well.	*A76286719: ...looking at your code you seem to lack a basic understanding of how you make Alloy models.*
Ability to guide LLM	13.04	Users need to use a good prompt or multiple rounds of dialogue for the LLM to generate properly usable code.	*A76009417: I was able to fix it with some ChatGPT assistance! I sent the error message to ChatGPT and it explained the following...I asked ChatGPT why is my nextjs code being run server-side?... so I asked ChatGPT can I make it so that an import only happens on the client side?... The first example didn't work out of the box, but with a little more questioning, I arrived at the solution.*
Hallucinate	7.25	LLM generates code that references nonexistent functions or libraries.	*Q76714882: I asked ChatGPT about cross-platform and it suggested using NAudio.Alsa for Linux. I think, it's hallucinating, since I don't see anything like that.*
Can't understand	7.25	Although usable code is generated, the user does not understand the meaning of the code.	*Q75839167: ChatGPT was right. But WHY? ChatGPTs response was...I have to be honest. I did not understand what ChatGPT tried to explain me here.*
Privacy doubt	5.80	Users are concerned about privacy being compromised when using LLM to generate code.	*Q72617988: I'm using the VS Code GitHub Copilot extension. Sometimes I edit files that contain secrets, and I don't want to accidentally send those to Microsoft-/GitHub.*

and the new platform also had specific SQL. The user encountered problems during the data migration process and did not know how to convert a large number of SOQL statements.

For traditional LCP, users needed to carefully choose a platform for their applications because each platform had its own domain-specific language and unique APIs. Once data and functionality were tightly integrated on a particular platform, migrating to another platform could require a complete application refactor, resulting in significant costs. This meant that users when selecting an LCP tool needed to balance not only current requirements but also consider potential changes and expansions in the future. In contrast, LLM-based LCP typically provided developers with more flexibility and control. Developers could manage data more freely, were no longer constrained by a specific platform; and thus, they did not need to worry about data migration issues between LCP platforms.

Finding 10: *Traditional LCP poses data migration challenges due to its closed nature, while LLM-based LCP offers greater flexibility in this regard.*

Uncertainty of LLM-based LCP. Users of LLM-based LCP faced limitations on "version problem", "ability to guide LLM" and "hallucinations", which reached 17.39%, 13.04% and 7.25%, respectively. These issues underscore the inherent uncertainty in using LLM-based LCP for programming and that software development did not guarantee satisfactory results. There was a risk of generating erroneous code or unsatisfactory solutions due to the hallucination problems of LLM or the developers' lack of ability to guide LLM.

To understand user satisfaction with using LLM-based LCP, we conducted a detailed analysis of the data related to LLM-based LCP in Section 4.1. In Section 4.1, we had already extracted 566 instances related to LLM-based LCP and discussed their distribution in application domains. We reanalyzed the aforementioned 566 instances to determine whether users provided positive feedback. For example, in *A75778070*, the user mentioned, *"...asked ChatGPT, it gave me this code which looks correct but just generates a Traceback"*, which was considered positive feedback. The judgment was made by the first author and verified by the second author.

The data presented in Table 8 illustrates the proportion of LLM-based LCP users who provided positive feedback across various application fields. The overall positive feedback rate stood at 39.22%. These findings suggest that users may not be highly satisfied with the solution generation using LLM-based LCP.

Overall, while LLM-based LCP offers innovative prospects, its positive feedback rate in practical applications remained modest. To enhance the stability of LLM-based LCP, developers and researchers needed to focus on and continuously improve above aspects to ensure that LLM-based LCP could better meet the needs of users.

Table 8: Positive feedback rate from LLM-based LCP users. PFR, PFI, and Ins. respectively represent positive feedback rate, positive feedback instances, and total instances.

Application Areas	PFR(%)	PFI	Ins.
General	43.01	78	186
Web Frontend	33.68	32	95
Web Backend	23.33	21	90
Database Management	48.72	38	78
Simple script	36.00	18	50
Algorithm	62.16	23	37
Game	23.08	3	13
Hardware	44.44	4	9
Others	62.50	5	8
Total	39.22	222	566

Finding 11: *Compared to traditional LCP, LLM-based LCP exhibits greater uncertainty, necessitating improvements to meet user expectations more reliably.*

5. DISCUSSION

Although traditional LCP and LLM-based LCP exhibited different characteristics (RQ1) and limitations (RQ3), there was a noticeable trend toward convergence in their key technologies. We first discussed how LLM/AI technology had recently been incorporated into traditional LCP tools in Section 5.1. Following that, in Section 5.2, we explored how visualization technology was used by recent LLM-based LCP tools. Section 5.3 was dedicated to examining the implications of 'LLM Agents' for LCP. Lastly, in Section 5.4, we discussed the potential threats to the validity of this study.

5.1. LLM/AI Technology Added to Traditional LCP Tools

Our analysis of website descriptions and documentation of traditional LCP tools presented in Section 3 revealed that 13 out of these 24 tools had incorporated AI technology since 2023. These functionalities were categorized into LLM-driven features and AI-driven features. LLM-driven features referred to the use of LLM in LCP tools to generate applications, perform data analysis, or generate workflows and code. On the other hand, AI-driven features referred to features driven by non-LLM AI, such as automatically recommending components that could be applied. By carefully reading their official documentation and experiencing the AI features of these LCP tools, we summarized several features as shown in Table 9.

5.1.1. App Generation Driven by LLM

Combined with LLM technology, users can generate applications using natural language in LCP tools without any manual or even visual programming. For example, in Quickbase [41], users can use natural language to describe things like, "*What do users want the app to do? Or what problem are users trying to*

Table 9: Recent AI-based features added to traditional LCP tools

LCP Tools	LLM-Driven			AI-Driven
	Generate Apps	**Data Analysis**	**Generate Workflow/Code**	**Recommend Field/Component**
Quickbase [41]	✓			✓
OutSystems [20]	✓	✓	✓	✓
Power Apps [8]	✓		✓	✓
Appypie [42]	✓			
Appsheet [43]	✓			
Softr [44]	✓			
Appian [45]		✓	✓	
Servicenow [46]		✓	✓	
Airtable [47]		✓	✓	✓
Mendix [48]				✓
Uipath [9]		✓	✓	
Pega [49]				✓
Nintex [50]				✓

solve? " Then, Quickbase can produce a fully functional application that meets their needs. After generating the application, users can make further detailed adjustments to it based on their requirements within the visual development interface. In SOFTR [44], users simply only need to select the application type and give prompt to describe the goals and functions of the application, then SOFTR can generate the application directly. This feature significantly saves time for programming development and can save time and learning costs for beginners. However, the generated applications by this technology tend to be simple and monotonous, making them better suited for straightforward functionalities, inspiration, and quick validation of ideas.

5.1.2. Workflow/Code Generation Driven by LLM

Users can generate workflows or code snippets using natural language directly in the LCP tool. Unlike generating applications using natural language mentioned earlier, this feature emphasizes more on the possibility of controllably generating or modifying parts of the application using natural language. For example, in APPIAN [45], users can generate a workflow of the complete process by talking to the built-in copilot and presenting it as a visualization module. This can then be reviewed and modified in a low-code visualization model in APPIAN. Similarly, in APPSHEET [43], developers talk to the built-in Duet AI to generate automated actions on the data. For example, "*Check if everything is working properly every week and submit a report.*" Then, Appsheet will automatically create forms to track the data. This feature enables a deep integration between traditional LCP and LLM-based LCP tools. Developers can not only generate applications using natural language but also exercise finer control over the generated applications on a visual canvas for meeting the specific requirements of their applications.

5.1.3. Data Analysis Driven by LLM

This feature refers to the ability for developers to utilize natural language to explore, summarize, analyze data, or automatically generate reports. For example, in AIRTABLE [47], customer service professionals can make natural language requests to AI to categorize their feedback data based on emotions such as positive, neutral, and negative. Subsequently, AI will autonomously perform this task and generate the corresponding table. In APPIAN, users can ask questions to the built-in LLM and seek specific insights into in-app data by using natural language. Integrating LLM into traditional LCP tools empowers them with data analysis capabilities. This feature not only allows developers to interact with data in a natural and intuitive way but also reduces the learning curve for developers who may not have a background in data science or complex query languages.

5.1.4. Field/Component Recommendation Driven by AI

Through learning user needs, project characteristics, and developer preferences, LCP tools can recommend the most suitable fields or components for the current context during the development process, thereby accelerating the development process. For example, when developing software in OUTSYSTEMS [20] platform, in the visual programming interface, logic flows are constructed by dragging and dropping components. OUTSYSTEMS will recommend to the developer the next possible components and the parameters of these components based on the current logic flow. This knowledge is derived from learning about 250,000 anonymous patterns in OUTSYSTEMS. This feature further speeds up the traditional LCP development workflow and reduces the cognitive load on the LCP tool for developers.

When investigating the AI-driven features added to traditional LCP tools, we observe an apparent trend that has emerged. The progressive integration of AI technologies within these platforms has served as a catalyst for heightened development efficiency and a reduction in the learning curve. Moreover, mainstream traditional LCP tools are actively exploring how to integrate LLM within the tool to maximize the advantages of LLM-based LCP and combine them with the strengths of traditional LCP.

5.2. Visualization drives LLM-based LCP

In the combination of LLM and visualization technology, in addition to the introduction of LLM in traditional LCP tools, there are some studies that use visualization technology to drive LLM-based LCP. Since LLMs at this stage are still not capable of aligning the generated code with the user's prompts, some researchers are attempting to bridge the gap by leveraging various visualization techniques.

5.2.1. LowCoder: Integrating Visual and Natural Language Interfaces for AI Pipeline Development.

LowCoder [51] is an LCP tool for developing AI pipelines that supports both a visual programming interface and an LLM-based natural language interface. Specifically, LowCoder uses visual programming as a read-write view and PBNL as a writing-only view, letting developers view data in a read-only view. The tool keeps these three views in sync by representing the program in a domain-specific language (DSL). Users can drag and drop any block of operators from the palette onto the canvas to form an AI pipeline, and set hyperparameters for each operator.

However, the tool palette contains more than a hundred blocks of operators, making it very difficult for the user to find the desired operator to use. LowCoder therefore provides an NL interface where the user can describe the desired operation in a text box and press the "Predict Pipeline" button. The tool then uses a natural language code-switching model to infer the relevant operators and any applicable hyperparameters, and automatically adds the most relevant operators to the end of the pipeline. In this tool, researchers use VPLs as the main approach to develop AI pipelines, supplemented by PBNL to view data. Additionally, employing LLM to recommend the next set of operators and parameters further accelerates development efficiency.

5.2.2. CoLadder: Enhanced Control Over LLM Code Generation with Visual Tools.

CoLadder [52] uses visualization to give developers control over the LLM code generation process. Specifically, CoLadder contains a tree-based prompt editor that allows developers to split tasks into manageable subtasks. This allows for more flexible construction of hierarchical prompt structures, resulting in aligned code structures that are not only easy to generate but also straightforward to verify.

In addition, the tool supports direct manipulation of the prompt structure through various prompt block operations. Developers can seamlessly add, edit, delete, or drag and drop prompt blocks via intuitive buttons. Each block-based operation triggers updates to the corresponding code and ensures that changes are propagated consistently to the rest of the code as needed. Finally, CoLadder utilizes a block-based design to modularize each prompt and snippet of code. After each module's prompt generates the code, the code can be modified flexibly in the block editor.

Overall, the design of CoLadder builds upon LLM-based LCP, supplemented with VPLs. This facilitates code generation and validation, providing developers with a high level of control and intuitiveness, thereby further reducing the threshold for developers to guide LLM.

5.2.3. Low-code LLM: Transforming Visual Workflows and Natural Language Descriptions.

Low-code LLM [53] is an LLM tool with visual interfaces for a wide variety of tasks, including programming. The user can first enter a short task description, and LLM will complete a prompted workflow in a specific format (with step-by-step descriptions and logical jumps) based on the task description, and visualize it to the user on the canvas. The canvas serves as an interactive space where developers have the flexibility to refine and customize the generated workflow through six distinct actions: add/remove steps, modify descriptions, add/remove a jump logic, drag and drop, extend sub-flowchart, and regenerate and confirm. Finally, after getting confirmation, the tool translates the specific format workflow into a natural language description and generates the code for the entire workflow by the LLM.

Unlike the previous two tools, this tool places more emphasis on using a specific format to transform visual workflows and natural language descriptions to each other, and directly inputs all generated natural language descriptions into LLM to generate code

5.3. LLM Agent for LCP

In Section 4, we found that developers primarily use LLM-based LCP during the implementation phase of development (Finding 4). Due to the hallucinations and uncertainties associated with LLMs, developers require specialized programming knowledge to design programming architectures or ensure code correctness (Finding 7). Some studies attempted to bridge this gap by employing agents based on LLM [27, 54, 26]. These agents could span the entire software development life cycle, reducing logical inconsistencies and errors in LLM. The following is a brief introduction to the three LLM agents to understand the impact of the agent on the LCP.

5.3.1. AutoGPT: LLM Single-Agent for Software Development

AutoGPT [54] is the most famous GPT agent project that can be used in software development, business analysis, market research, and other fields. As of the time of writing this paper, there are 155k stars on GitHub. Developers only need to provide a prompt or a set of natural language instructions. AutoGPT decomposes the target into subtasks, which are then linked together and executed in order to produce larger results initially arranged by user input. In the field of software engineering, AutoGPT can assign different roles to intelligent agents, thereby autonomously executing the capabilities of different stages in the software development life cycle. However, due to the fact that AutoGPT did not optimize collaboration between different agents, this tool still has hallucinations and is difficult to solve complex engineering problems.

5.3.2. MetaGPT: LLM Multi-Agent for Software Development

In MetaGPT [27], five roles are defined that span the entire software development cycle: Product Manager, Architect, Project Manager, Engineer, and QA Engineer, and they are assigned features, goals, and constraints. All agents follow the software development SOP, such as the Product Manager conducting in-depth analysis of user requirements, developing product requirements documents, and then handing them over to the Architect. In addition, Engineer and QA Engineer can communicate with each other to correct any errors that may exist in the generated code. This tool effectively decomposes complex software engineering tasks into subtasks involving collaboration between multiple agents, allowing more complex software to be developed.

5.3.3. ChatDev: LLM Multi-Agent for Software Development Based on Chat Chain

This tool [26] divides the development process into four stages: design, coding, testing, and documentation. At each stage, ChatDev recruits multiple agents with different roles, such as programmers, reviewers, and testers. Through the chat chain, each stage is divided into atomic subtasks, and two adjacent roles in the chat chain participate in context aware multi round discussions to propose and validate solutions. Unlike the MetaGPT, the proposed chat chain of this tool collaborates through natural language. This multi-agent assisted system also spans the entire software development cycle, alleviating the hallucinations of LLM and providing inspiration for future low-code development.

In summary, the concept of an LLM Agent for LCP provides a different solution than combining visualization to solve the complexity and unreliability of LLM in the software development life cycle. This is via collaborative communication among multiple agents, providing valuable insights for future development of LCP.

5.4. Threats to Validity

Internal Validity: Key internal threats are that authors might be biased when performing data analysis. To reduce subjective biases and enhance the dependability of our findings, we adopted a dual-author approach for each analysis, guaranteeing comprehensive resolution of any disagreements that arose. We measured inter-coder agreement, yielding a Cohen's kappa of 0.89, indicating strong consistency between annotators.

In addition, when constructing the LCP dataset based on LLMs, we adopted a two-stage process of keyword prescreening and ChatGPT semantic filtering. Since the second stage may introduce bias, we carefully designed and standardized the prompts. This design was informed by prior work on prompt engineering to ensure a higher recall rate (86.7%). To control potential false positives resulting from the high recall, all filtered posts underwent a second round of manual verification to ensure overall data quality. At the same time, we acknowledge the non-determinism of large language models; in future work, we plan to explore more robust approaches, such as model fine-tuning and ensemble/multi-agent frameworks, to further enhance reproducibility and stability.

External Validity: Key external threats are related to the generalizability of our research findings. Our study is based on discussion data from developers on SO. However, since LLM-based LCP is a relatively recent development and is still evolving and changing rapidly, discussions on SO may not necessarily represent the latest state of affairs. Additionally, while SO is a leading platform for developer Q&A, it may not fully represent the diverse discussions occurring in the broader programming community. In particular, discussions of traditional LCP on SO may overlook its adoption in enterprise or non-developer contexts, where such platforms are widely used but less often discussed in public forums. To ensure high-quality discussions, we manually evaluated data by selecting posts based on their scores, from high to low. We explicitly acknowledge this single-source reliance as a threat to validity. While this study focuses on Stack Overflow data, future work will extend the analysis by triangulating these findings with supplementary sources such as technical blogs, and GitHub discussions.

6. Related Work

6.1. Low-Code Programming

With the growing popularity of low-code approaches, LCP has been applied in various fields such as human resources, the Internet of Things, machine learning, etc [55, 56, 57, 58]. Zhuang et al. [59] proposed a low-code federated learning framework. By simplifying APIs and adopting a modular design, this framework enables users with varying levels of expertise to experiment with federated learning applications with minimal coding. Chen et al. [60] proposed DeviceTalk, a low-code IoT development framework that utilizes visual techniques to accelerate software development in the field of distributed intelligent system devices.

Because the concept of low-code is relatively vague, some studies have explored the concept and models of low-code. Di Ruscio et al. [18] compared low-code with model-driven approaches, identified their similarities and differences, and analyzed the strengths and weaknesses of these two methods. Gomes and Brito [61] conducted a descriptive study on low-code, exploring and showcasing key concepts, factors, and variables related to low-code development platforms. Hirzel [2] integrated low-code literature from various research fields, explained the technical principles of low-code programming, and provided a unified perspective on it.

In addition, some studies have empirically investigated LCP from the perspective of developers. Rafi et al. [62] conducted interviews with developers using low-code to understand practitioners' views on the low-code trend. This research found that the emergence of low-code can greatly facilitate the development of high-quality products with low cost and time. Luo et al. [63] used data from SO and Reddit to analyze the descriptions, advantages, limitations, and challenges of low-code development. Al Alamin et al. [31] conducted empirical research on SO data. This research used topic modeling to analyze the distribution of discussion topics on SO, thus identifying the difficulties users face in the process of using LCP.

There are some studies that summarize the characteristics and challenges of LCP through literature reviews. Sufi [64] conducted an extensive literature review on topics such as low-code, no-code, visual programming, and model-driven programming. The review explored the advantages, limitations, features, and application areas of LCP. Rokis and Kirikova [65] carried out a literature review to systematically summarize various challenges in low-code software development.

Although the above-mentioned papers have explored the concepts, characteristics, and challenges of LCP using various methods, these studies have confined the concept of LCP solely to tools related to visualization and modularity, namely the traditional LCP proposed in this paper. The characteristics, application domains, and limitations faced by LLM-based LCP are distinct from traditional LCP. Thus, the latest LLM-based LCP requires further research.

6.2. Large Language Models for Software Engineering

LLM have garnered considerable attention in academic and industrial circles, showcasing remarkable proficiency in various tasks across multiple disciplines[12]. Notably, they have the ability to generate usable code, providing invaluable support to LCP [66, 67, 68]. In the field of software development, tools such as GPT4 and Copilot can generate code based on the user's instructions [69, 70, 71, 72]. Users can talk to a LLM, describe their task, and let it generate code to complete the task. Alternatively, give the starting code snippet and let it continue. LLM increases the efficiency of software developers and helps amateur developers develop.

Several studies investigated the code generation ability of LLM and its applications in the field of software engineering. Liang et al. [73] surveyed the users of AI Programming Assistants to understand how developers use these tools and the usability challenges they face.

Sridhara et al. [74] examined the utilization of ChatGPT in executing various common software engineering tasks and compared and analyzed using ground facts from human experts. Poldrack et al. [11] studied the ability to complete programming tasks and the quality of generated code using GPT4. Bubeck et al. [12] demonstrated in a set of coding challenges that GPT4's coding capabilities achieved human-level performance. Suri et al. [75] delved into the capabilities and challenges of auto-GPT in software engineering practices, especially dealing with complex frameworks such as Sping Boot, Django, and Flask.

The above studies investigated the programming capabilities of LLMs and agents based on LLMs, as well as their ability to perform software engineering tasks. These studies also examined the challenges faced by developers applying LLMs. However, the literature mentioned above did not explore the characteristics and limitations of LLMs from the perspective of LCP. Our study explores the similarities and differences between LLM-based LCP and traditional LCP, discussing how LLM-based LCP and traditional LCP can be integrated.

7. Conclusion and Future work

In this paper, we delve into the similarities and differences between traditional LCP and LLM-based LCP. Our empirical study involved collecting related posts on SO over the past three years to determine the similarities and differences. Our findings highlight commonalities and disparities in the development areas, limitations, and distribution of software development cycles. They are mainly used in the field of web development and face limitations such as "Reliablity doubt" and "Need for professional knowledge". Their discussions are similarly distributed throughout the software development cycle. However, LLM-based LCP applies to a wider range of scenarios and solves more problems, facing problems specific to larger language models such as "Hallucinate". In discussions of the software development cycle, there is also more discussion of deployment issues. To the best of our knowledge, this is the first empirical study of LLM-based LCP. This research can help developers of low-code tools to better understand and integrate the two.

In future work, we will extend our analysis beyond SO by incorporating diverse data sources such as GitHub Discussions and technical blogs, and applying triangulation to mitigate single-source bias. This will enable cross-platform comparisons in terms of question types, application domains, and tool usage, thereby enhancing the robustness of our conclusions. In addition, we plan to conduct user studies, such as surveys and interviews with practitioners of low-code tools, to gather direct feedback on their experiences with LLM-based and traditional LCPs, thereby validating and refining our findings.

8. Conflict of Interests

The authors declared that they have no conflict of interest exits in the submission of this manuscript, and manuscript is approved by all authors for publication. I would like to declare on behalf of my co-authors that the work described was original research that has not been published previously and is not under consideration for publication elsewhere. All the authors listed have approved the manuscript.

We declare that we do not have any commercial or associative interest that represents a conflict of interest in connection with the work submitted.

9. Acknowledgements

This work is supported by the National Natural Science Foundation of China (No. 62332004), the National Natural Science Foundation of China (62276279, 62302534), Guangdong Basic and Applied Basic Research Foundation (2024B1515020032, 2025A1515011632) and the National Key Research and Development Program of China (2023YFB2703700).

10. Data availability statements

The dataset and results of this paper are open source and available in `https://zenodo.org/records/11232842`.

References

[1] A. C. Bock and U. Frank, ‘‘Low-code platform,’’ Business & Information Systems Engineering, vol. 63, pp. 733--740, 2021.

[2] M. Hirzel, ‘‘Low-code programming models,’’ Communications of the ACM, vol. 66, no. 10, pp. 76--85, 2023.

[3] A. Sahay, A. Indamutsa, D. Di Ruscio, and A. Pierantonio, ‘‘Supporting the understanding and comparison of low-code development platforms,’’ in 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE, 2020, pp. 171--178.

[4] A. Bragança, I. Azevedo, N. Bettencourt, C. Morais, D. Teixeira, and D. Caetano, ‘‘Towards supporting spl engineering in low-code platforms using a dsl approach,’’ in Proceedings of the 20th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences, 2021, pp. 16--28.

[5] M. M. Burnett and D. W. McIntyre, ‘‘Visual programming,’’ COmputer-Los Alamitos-, vol. 28, pp. 14--14, 1995.

[6] A. Cypher and D. C. Halbert, Watch what I do: programming by demonstration. MIT press, 1993.

[7] E. Jiang, E. Toh, A. Molina, K. Olson, C. Kayacik, A. Donsbach, C. J. Cai, and M. Terry, ‘‘Discovering the syntax and strategies of natural language programming with generative language models,’’ in Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems, 2022, pp. 1--19.

[8] Microsoft. (2023) Power apps. [Online]. Available: https://powerapps.microsoft.com

[9] U. Team. (2023) Uipath. [Online]. Available: https://www.uipath.com

[10] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell et al., ‘‘Language models are few-shot learners,’’ Advances in neural information processing systems, vol. 33, pp. 1877--1901, 2020.

[11] R. A. Poldrack, T. Lu, and G. Beguš, ‘‘Ai-assisted coding: Experiments with gpt-4,’’ arXiv preprint arXiv:2304.13187, 2023.

[12] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg et al., ‘‘Sparks of artificial general intelligence: Early experiments with gpt-4,’’ arXiv preprint arXiv:2303.12712, 2023.

[13] S. O. Team. (2023) Stack overflow. [Online]. Available: https://stackoverflow.com

[14] B. G. Glaser, ‘‘The constant comparative method of qualitative analysis,’’ Social problems, vol. 12, no. 4, pp. 436--445, 1965.

[15] R. Clay, R. John, R, M. Christopher, C. Alex, and W. Dominique, ‘‘Vendor landscape: A fork in the road for low-code development platforms,’’ Analyst Report. Forrester Research Inc, 2014.

[16] C. Richardson and J. R. Rymer, ‘‘The forrester wave™: low-code development platforms, q2 2016,’’ Forrester, Washington DC, 2016.

[17] J. R. Rymer et al., ‘‘Vendor landscape: A fork in the road for low-code development platforms,’’ Analyst Report. Forrester Research Inc, 2017.

[18] D. Di Ruscio, D. Kolovos, J. de Lara, A. Pierantonio, M. Tisi, and M. Wimmer, ‘‘Low-code development and model-driven engineering: Two sides of the same coin?’’ Software and Systems Modeling, vol. 21, no. 2, pp. 437--446, 2022.

[19] W. M. Van der Aalst, M. Bichler, and A. Heinzl, ‘‘Robotic process automation,’’ pp. 269--272, 2018.

[20] O. Team. (2023) Outsystems. [Online]. Available: https://www.outsystems.com/

[21] Github. (2023) Github copilot. [Online]. Available: https://github.com/features/copilot

[22] OpenAI. (2023) Chatgpt. [Online]. Available: https://openai.com/chatgpt

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, . Kaiser, and I. Polosukhin, ‘‘Attention is all you need,’’ Advances in neural information processing systems, vol. 30, 2017.

[24] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, ‘‘Large language models for software engineering: A systematic literature review,’’ arXiv preprint arXiv:2308.10620, 2023.

[25] W. Ma, S. Liu, W. Wang, Q. Hu, Y. Liu, C. Zhang, L. Nie, and Y. Liu, ‘‘The scope of chatgpt in software engineering: A thorough investigation,’’ arXiv preprint arXiv:2305.12138, 2023.

[26] C. Qian, X. Cong, W. Liu, C. Yang, W. Chen, Y. Su, Y. Dang, J. Li, J. Xu, D. Li, Z. Liu, and M. Sun, ‘‘Communicative agents for software development,’’ 2023.

[27] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, C. Zhang, J. Wang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber, ‘‘Metagpt: Meta programming for a multi-agent collaborative framework,’’ 2023.

[28] J. Zamfirescu-Pereira, R. Y. Wong, B. Hartmann, and Q. Yang, ‘‘Why johnny can’t prompt: how non-ai experts try (and fail) to design llm prompts,’’ in Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, 2023, pp. 1--21.

[29] S. Shylesh, ‘‘A study of software development life cycle process models,’’ in National Conference on Reinventing Opportunities in Management, IT, and Social Sciences, 2017, pp. 534--541.

[30] S. E. Community. (2023) Stack overflow data dump. [Online]. Available: https://archive.org/details/stackexchange

[31] M. A. Al Alamin, S. Malakar, G. Uddin, S. Afroz, T. B. Haider, and A. Iqbal, ‘‘An empirical study of developer discussions on low-code software development challenges,’’ in 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR). IEEE, 2021, pp. 46--57.

[32] G. Team. (2023) G2.com. [Online]. Available: https://www.g2.com/

[33] M. U. Hadi, R. Qureshi, A. Shah, M. Irfan, A. Zafar, M. Shaikh, N. Akhtar, J. Wu, and S. Mirjalili, ‘‘A survey on large language models: Applications, challenges, limitations, and practical usage,’’ TechRxiv, 2023.

[34] T. Teubner, C. M. Flath, C. Weinhardt, W. van der Aalst, and O. Hinz, ‘‘Welcome to the era of chatgpt et al. the prospects of large language models,’’ Business & Information Systems Engineering, vol. 65, no. 2, pp. 95--101, 2023.

[35] G. Orrù, A. Piarulli, C. Conversano, and A. Gemignani, ‘‘Human-like problem-solving abilities in large language models using chatgpt,’’ Frontiers in Artificial Intelligence, vol. 6, p. 1199350, 2023.

[36] J. S. Park, J. C. O’Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, ‘‘Generative agents: Interactive simulacra of human behavior,’’ arXiv preprint arXiv:2304.03442, 2023.

[37] B. Xu, A. Yang, J. Lin, Q. Wang, C. Zhou, Y. Zhang, and Z. Mao, ‘‘Expertprompting: Instructing large language models to be distinguished experts,’’ arXiv preprint arXiv:2305.14688, 2023.

[38] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou et al., ‘‘Chain-of-thought prompting elicits reasoning in large language

models,'' Advances in Neural Information Processing Systems, vol. 35, pp. 24 824--24 837, 2022.
[39] M. Team. (2023) Maxqda. [Online]. Available: https://www.maxqda.com/
[40] J. Community. (2023) The state of developer ecosystem 2022. [Online]. Available: https://www.jetbrains.com/lp/devecosystem-2022/
[41] Q. Team. (2023) Quickbase. [Online]. Available: https://www.quickbase.com/
[42] A. Team. (2023) Appypie. [Online]. Available: https://www.appypie.com/
[43] ------. (2023) Appsheet. [Online]. Available: https://www.appsheet.com/
[44] S. Team. (2023) Softr. [Online]. Available: https://www.softr.io/
[45] A. Team. (2023) Appian. [Online]. Available: https://appian.com/
[46] S. Team. (2023) Servicenow. [Online]. Available: https://www.servicenow.com/
[47] A. Team. (2023) Airtable. [Online]. Available: https://www.airtable.com/
[48] M. Team. (2023) Mendix. [Online]. Available: https://www.mendix.com/
[49] P. Team. (2023) Pega. [Online]. Available: https://www.pega.com/
[50] N. Team. (2023) Nintex. [Online]. Available: https://www.nintex.com/
[51] N. Rao, J. Tsay, K. Kate, V. J. Hellendoorn, and M. Hirzel, ''Ai for low-code for ai,'' arXiv preprint arXiv:2305.20015, 2023.
[52] R. Yen, J. Zhu, S. Suh, H. Xia, and J. Zhao, ''Coladder: Supporting programmers with hierarchical code generation in multi-level abstraction,'' arXiv preprint arXiv:2310.08699, 2023.
[53] Y. Cai, S. Mao, W. Wu, Z. Wang, Y. Liang, T. Ge, C. Wu, W. You, T. Song, Y. Xia et al., ''Low-code llm: Visual programming over llms,'' arXiv preprint arXiv:2304.08103, 2023.
[54] T. et al. (2023) Autogpt. [Online]. Available: https://github.com/Significant-Gravitas/AutoGPT
[55] C. Di Sipio, D. Di Ruscio, and P. T. Nguyen, ''Democratizing the development of recommender systems by means of low-code platforms,'' in Proceedings of the 23rd ACM/IEEE international conference on model driven engineering languages and systems: companion proceedings, 2020, pp. 1--9.
[56] F. Ihirwe, D. Di Ruscio, S. Mazzini, P. Pierini, and A. Pierantonio, ''Low-code engineering for internet of things: a state of research,'' in Proceedings of the 23rd ACM/IEEE international conference on model driven engineering languages and systems: companion proceedings, 2020, pp. 1--8.
[57] I. N. Oteyo, A. L. S. Pupo, J. Zaman, S. Kimani, W. De Meuter, and E. G. Boix, ''Building smart agriculture applications using low-code tools: the case for discopar,'' in 2021 IEEE AFRICON. IEEE, 2021, pp. 1--6.
[58] R. Martins, F. Caldeira, F. Sa, M. Abbasi, and P. Martins, ''An overview on how to develop a low-code application using outsystems,'' in 2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE). IEEE, 2020, pp. 395--401.
[59] W. Zhuang, X. Gan, Y. Wen, and S. Zhang, ''Easyfl: A low-code federated learning platform for dummies,'' IEEE Internet of Things Journal, vol. 9, no. 15, pp. 13 740--13 754, 2022.
[60] W.-E. Chen, Y.-B. Lin, T.-H. Yen, S.-R. Peng, and Y.-W. Lin, ''Devicetalk: A no-code low-code iot device code generation,'' Sensors, vol. 22, no. 13, p. 4942, 2022.
[61] P. M. Gomes and M. A. Brito, ''Low-code development platforms: a descriptive study,'' in 2022 17th Iberian Conference on Information Systems and Technologies (CISTI). IEEE, 2022, pp. 1--4.
[62] S. Rafi, M. A. Akbar, M. Sánchez-Gordón, and R. Colomo-Palacios, ''Devops practitioners' perceptions of the low-code trend,'' in Proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2022, pp. 301--306.
[63] Y. Luo, P. Liang, C. Wang, M. Shahin, and J. Zhan, ''Characteristics and challenges of low-code development: the practitioners' perspective,'' in Proceedings of the 15th ACM/IEEE international symposium on empirical software engineering and measurement (ESEM), 2021, pp. 1--11.
[64] F. Sufi, ''Algorithms in low-code-no-code for research applications: a practical review,'' Algorithms, vol. 16, no. 2, p. 108, 2023.
[65] K. Rokis and M. Kirikova, ''Challenges of low-code/no-code software development: A literature review,'' in International Conference on Business Informatics Research. Springer, 2022, pp. 3--17.
[66] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman et al., ''Evaluating large language models trained on code,'' arXiv preprint arXiv:2107.03374, 2021.
[67] S. I. Ross, F. Martinez, S. Houde, M. Muller, and J. D. Weisz, ''The programmer's assistant: Conversational interaction with a large language model for software development,'' in Proceedings of the 28th International Conference on Intelligent User Interfaces, 2023, pp. 491--514.
[68] M. X. Liu, A. Sarkar, C. Negreanu, B. Zorn, J. Williams, N. Toronto, and A. D. Gordon, ''\what it wants me to say": Bridging the abstraction gap between end-user programmers and code-generating large language models,'' in Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, 2023, pp. 1--31.
[69] M. Kazemitabaar, J. Chow, C. K. T. Ma, B. J. Ericson, D. Weintrop, and T. Grossman, ''Studying the effect of ai code generators on supporting novice learners in introductory programming,'' in Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, 2023, pp. 1--23.
[70] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, ''The impact of ai on developer productivity: Evidence from github copilot,'' arXiv preprint arXiv:2302.06590, 2023.
[71] N. Nguyen and S. Nadi, ''An empirical evaluation of github copilot's code suggestions,'' in Proceedings of the 19th International Conference on Mining Software Repositories, 2022, pp. 1--5.
[72] P. Vaithilingam, T. Zhang, and E. L. Glassman, ''Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models,'' in Chi conference on human factors in computing systems extended abstracts, 2022, pp. 1--7.
[73] J. T. Liang, C. Yang, and B. A. Myers, ''A large-scale survey on the usability of ai programming assistants: Successes and challenges,'' in 2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE). IEEE Computer Society, 2023, pp. 605--617.
[74] G. Sridhara, S. Mazumdar et al., ''Chatgpt: A study on its utility for ubiquitous software engineering tasks,'' arXiv preprint arXiv:2305.16837, 2023.
[75] S. Suri, S. N. Das, K. Singi, K. Dey, V. S. Sharma, and V. Kaulgud, ''Software engineering using

autonomous agents: Are we there yet?'' in 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2023, pp. 1855--1857.