AI-Augmented Low-Code Development: Bridging the Gap Between Citizen Developers and Software Engineers

Mr. Harsh Virani (Mtr. Nr. - 10030225)
MSc Artificial Intelligence & Robotics
Hof University of Applied Sciences
Hof, Germany
harsh.virani@hof-univeristy.de

***Abstract*—The increasing demand for software necessitates engaging diverse developer groups, including Citizen Developers, to contribute alongside professional Software Engineers. Low-Code/No-Code (LCNC) platforms have emerged to simplify development processes. Concurrently, Artificial Intelligence (AI) is being applied with growing frequency across various aspects of software engineering (SE). This paper presents a review of the literature focusing on the emerging field of AI-augmented LCNC development, which holds promise for bridging the development gap between Citizen Developers and professional Software Engineers. We synthesize research concerning the application of AI in the SE domain and specific AI capabilities, particularly those drawing on Large Language Models (LLMs) and LLM-based agents, that are being integrated into LCNC platforms. The review highlights reported benefits, which include improvements in efficiency across multiple dimensions such as time, effort, collaboration, and financial costs, as well as the potential for fostering enhanced collaboration between different developer types. Furthermore, the discussion covers significant challenges encountered in this area, including those related to user skills, technical constraints, and organizational factors. Drawing from this synthesis, we identify critical research gaps and outline future directions for the AI-augmented LCNC field. These include the need for further investigation into task-technology fit, the refinement of performance evaluation metrics, and the exploration of proactive AI assistance mechanisms. This comprehensive review aims to provide researchers and practitioners with a structured overview of the current state, potential, and key challenges within AI-augmented LCNC development, serving as a foundation for future work.**

***Index Terms*—Large Language Models, Low-code development, Citizen development, AI augmentation, Bridging the Gap**

I. Introduction

The increasing demand for software solutions and the inherent limitations of traditional software development methods are significantly impacting the current technological landscape. This pressure necessitates alternative approaches that can deliver applications more rapidly and involve a broader range of contributors. Low-Code/No-Code (LCNC) development has emerged as a response to these challenges, offering platforms that enable the creation of software applications with minimal or no traditional coding. LCNC technologies typically employ visual programming and simplified interfaces to democratize software creation, empowering individuals often referred to as Citizen Developers (CDs). Low-code technologies are increasingly gaining importance in various contexts. Simultaneously, Artificial Intelligence (AI), particularly advancements in Large Language Models (LLMs), is being integrated into traditional software engineering (SE) practices. AI is utilized across the Software Development Lifecycle (SDLC) for tasks like code generation and providing intelligent recommendations to professional developers.

A notable distinction exists between the skill sets and roles of professional software engineers and citizen developers. Citizen developers frequently encounter difficulties in navigating the complexities of traditional programming. While LCNC platforms effectively enable CDs, and AI tools improve the efficiency of SEs, the integration of AI capabilities directly into LCNC platforms presents a significant and promising opportunity. This combination holds the potential to diminish the technical gap between CDs and SEs and enhance collaboration between these differing developer groups. However, the precise implications, benefits, challenges, and the impact on the CD-SE dynamic introduced by AI augmentation within LCNC platforms bring about new complexities and research questions. Investigating these aspects systematically is necessary. Furthermore, there is currently a limited amount of literature and practical experience regarding the use of generative AI specifically within low-code development, highlighting that this area is still relatively new for research.

Therefore, the main goal of this paper is to conduct a comprehensive review of the existing literature concerning the intersection of AI and LCNC development and its potential role in bridging the technical divide between Citizen Developers and Software Engineers. This review aims to characterize the current state of AI integration in LCNC platforms; identify the reported benefits of AI-augmented LCNC for both citizen developers and professional software engineers; analyze the associated challenges and limitations; understand how AI-augmented LCNC influences the roles and interaction dynamics between CDs and SEs as discussed in the literature; and propose a concise agenda for future research in this interdisciplinary domain. This involves synthesizing

the current research landscape and identifying areas requiring further exploration.

The subsequent sections of this paper are organized as follows: Section 2 will provide necessary background on Low-Code/No-Code development and the role of AI in Software Engineering, along with distinguishing the characteristics of Citizen Developers and Software Engineers. Section 3 will delve into the mechanisms and specific capabilities of AI integration within LCNC platforms. Section 4 will synthesize the benefits and empirical evidence regarding AI-augmented LCNC for different developer profiles. Section 5 will discuss the identified challenges and limitations. Section 6 will outline research gaps and suggest future research directions. Finally, Section 7 will conclude the paper by summarizing the key findings and their implications.

II. Background and Related Work

This section provides essential background information on Low-Code/No-Code (LCNC) development, the role of Artificial Intelligence (AI) in software engineering, the characteristics of citizen developers and software engineers, and the rationale for bridging the gap between these developer profiles.

A. Low-Code/No-Code Development Platforms

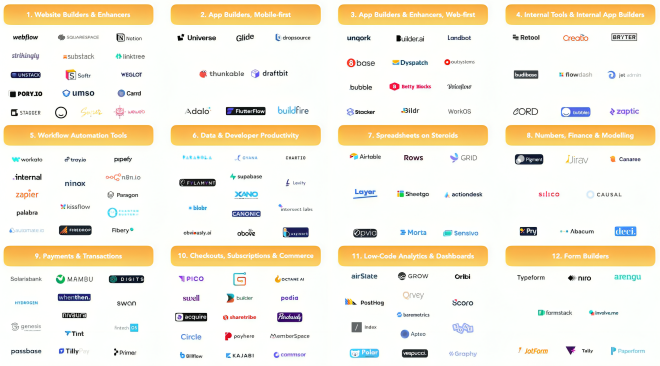

Fig. 1. LCNC platforms for various domains

Low-Code Development Platforms (LCDPs) enable application development with minimal or no coding by utilizing prebuilt components and visual interfaces. LCNC development allows for system creation using simplified tools, reducing the need for extensive programming knowledge. The fundamental concept of low code is not new, but it has gained increasing importance in various contexts within a platform-driven software landscape, with new features emerging, including those influenced by generative AI. LCDPs are designed to democratize software development by empowering individuals, including those without prior programming experience, who are often referred to as "citizen developers". While some studies suggest LCDPs are better suited for those with minimal coding skills, others consider their suitability for users with no coding skills, highlighting a need for clearer definitions. These platforms leverage simple, convenient, drag-and-drop visual layers to facilitate custom changes and prototyping for business applications. The simplified design also contributes to the ease of maintenance for applications developed on LCNC platforms, as they require little or no code for updates, enabling instant changes in response to evolving business requirements.

B. Role of AI in Software Engineering

Artificial Intelligence is profoundly transforming the field of software engineering, integrating into various stages of the software development lifecycle. This includes applications in requirement engineering, where AI can assist with tasks like classification and generation, code generation and software development, debugging, and code evaluation. AI-assisted code generators, powered by advancements in machine learning and natural language processing, can produce code from natural language descriptions. Examples of such tools include ChatGPT, OpenAI Codex, Cursor AI and GitHub Copilot [1]. LLMs are being applied to help programmers generate code and fix bugs. Their capacity to understand and complete code and text is based on input, training data, and reasoning abilities. The performance and quality of AI-generated code can be evaluated using metrics such as Code Validity, Correctness (assessed by passing unit tests), Security, Maintainability (which reveals issues like improper naming and high cognitive complexity), and Reliability. Studies indicate that the quality of the input, including detailed prompts and comments, significantly influences the outcome [1].

C. Characterizing Citizen Developers and Software Engineers

The literature distinguishes between different profiles involved in software creation, primarily citizen developers (CDs) and professional software engineers (SEs). Citizen developers are typically defined as individuals who lack extensive formal programming training and utilize LCNC platforms to build applications. Professional software engineers, conversely, possess deep technical programming expertise and are primarily engaged in traditional software development processes. Understanding the distinct characteristics, skill sets, and roles of these two groups is crucial for effective LCNC adoption and deployment within organizations. An empirical study defined experienced participants based on their familiarity with multiple programming languages/frameworks and self-reported moderate to high development experience, while inexperienced participants had no prior programming or LCNC platform use [2]. Experience, even for citizen developers using user-friendly platforms like PowerApps, influences the time needed to gain insight into the platform's capabilities, although such tools aim to simplify the process for both technical and non-technical users [3].

D. The Need to Bridge the Gap

Effectively leveraging the full potential of LCNC development necessitates bridging the gap between citizen developers and professional software engineers [4]. A key concern is the potential for programs developed by users lacking technical depth to suffer from poor quality or introduce long-term

issues. Consequently, there is an identified need to foster closer collaboration and linkage between citizen developers and professional developers [5]. Empirical studies that compare the performance of individuals with varying programming backgrounds on LCNC platforms are valuable. Such studies can help identify the levels of task complexity at which experienced developers demonstrate an advantage and the degree of tool complexity that can be effectively managed by inexperienced users [2]. This understanding is vital for informing management decisions regarding task allocation, assignment strategies, and the selection of appropriate LCNC tools for different user groups. While a study found comparable performance on simple tasks using a basic no-code builder regardless of prior experience, suggesting training concerns might be overstated for such cases, further research is needed to explore more complex tasks and industrial-scale platforms [2].

III. AI Augmentation in Low-Code Platforms

This section explores how Artificial Intelligence capabilities are being integrated into Low-Code/No-Code (LCNC) platforms and the specific AI-powered features that are emerging as a result.

A. Mechanisms of AI Integration

Artificial Intelligence is recognized as a crucial player in the evolution of low-code and no-code platforms in 2025. AI is taking these platforms to the next level by complementing existing functionalities and making them even simpler to use. The potential of generative AI for low code development platforms has been discovered in various ways, with new generative AI features increasingly announced [5]. While code generators have primarily supported conventional software development, their use is extending to power AI systems and integrate with user-friendly No-Code/Low-Code assistance systems. This integration aims to make AI accessible to non-AI experts. AI features, such as Natural Language Processing (NLP), machine learning-based code suggestions, and smart templates, allow even non-technical users to build sophisticated applications [3]. This signifies a paradigm shift towards AI-assisted low-code and no-code platforms focused on inclusivity, efficiency, and innovation, blurring the line between technical and non-technical users [3]. Integrating AI into these platforms is transforming application engagement and enabling organizations to rethink software development possibilities [3]. AI is fundamentally changing how new applications are developed, empowering non-developers and accelerating automation.

B. Key AI-Powered Capabilities

AI augmentation is introducing specific capabilities into LCNC platforms to enhance their functionality and usability.

Natural Language Interfaces (NLIs) and Programming by Natural Language (PBNL): A key area where AI has been an important enabler for LCNC platforms is in the understanding of human intent using natural language. Users can describe their needs in simple plain language, and AI translates it into working application logic or components. This allows users to articulate requirements in everyday language, and the platform generates the necessary application pieces. This serves as a bridge between technical skills and business solution creativity, enabling faster prototyping and deployment. It also provides a way to access large APIs without needing to memorize documentation. NLIs act as an intelligent search tool within the platform, helping users find desired functionality.

Intelligent Search and Discovery: AI assists users in finding the right operators, components, or functions within the LCNC platform. This enhanced discoverability, particularly through Natural Language interfaces, makes it easier for users to locate needed elements.

Code Generation and Assistance: AI can generate code snippets or logic components, even within environments designed for minimal or no coding. Tools based on Large Language Models (LLMs) can autonomously create programs, classes, functions, and test cases in various programming languages, enabling the translation of natural language into executable code [6]. GPT and other AI models help users create application components in a ready-made programming language described in everyday human language. AI code generators have established themselves as powerful tools that can accelerate and automate the development of conventional software [7]. They can generate program code based on prompts, deriving knowledge from previous projects.

Automation of Development Tasks: AI integration increases productivity by automating repetitive tasks within the LCNC workflow. This includes tasks such as data integration, workflow creation, and testing [3]. AI can automate testing, quickly identifying errors or performance bottlenecks. Beyond LCNC, AI in software engineering is broadly used for automating tasks like testing and deployment processes, and developers anticipate AI alleviating the tedium of routine tasks such as generating tests and documentation. AI can also assist with performing root-cause analysis for bugs and incidents, automating data analysis and pattern recognition.

AI-Enhanced User Interfaces: AI contributes to improved user experience through features like responsive templates and generally more user-friendly interfaces for no-code development. Machine-learning algorithms can examine user behavior and app data to suggest optimal design elements or workflows [3]. User-friendly graphical interfaces, combined with AI assistants, are essential for making no-/low-code AI platforms accessible, especially for inexperienced users.

C. Examples

Examples illustrate how AI augmentation is being implemented in practice, particularly through advanced code generation tools that demonstrate capabilities like Natural Language Interfaces (NLIs), Programming by Natural Language (PBNL), Intelligent Search, and Code Generation/Assistance.

GitHub Copilot: As an AI pair programmer, GitHub Copilot is a prominent example of AI augmenting development. It utilizes the OpenAI Codex Model, which relies on GPT models

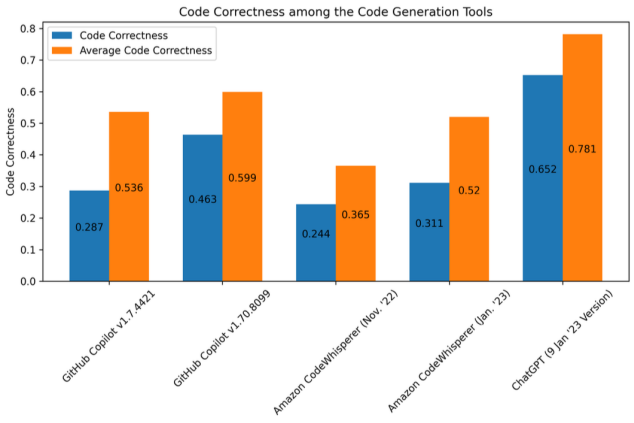

Fig. 2. Code Correctness and Average Code Correctness Scores of the Code Generation Tools [1].

fine-tuned on public code from GitHub. Copilot operates as an extension within popular IDEs like Visual Studio Code and JetBrains IDEs, offering real-time code suggestions and completion based on user code samples and context. It can generate code from natural language prompts or partial code inputs. Users can interact with it through different modes, including an "acceleration mode" for quickly solving known problems and an "exploration mode" for exploring ideas. Studies indicate it can significantly enhance developer productivity, with users accepting approximately 30% of suggested code on average [7]. GitHub Copilot can generate code in bulk or line-by-line. It is noted as one of the pioneers in the field of AI-assisted code generation. While powerful, generated code can contain security vulnerabilities.

Amazon CodeWhisperer: This tool enhances developer productivity by generating code recommendations based on developers' comments in English and prior code in the IDE [1]. It leverages ML models trained on various data sources, including Amazon's and open-source code. CodeWhisperer provides code snippets directly within the editor and is specifically designed to simplify the use of AWS services by suggesting AWS API code. It supports multiple IDEs and programming languages like Java, JavaScript, Python, C#, and Typescript [1]. CodeWhisperer has a unique feature of a reference tracker that identifies similarities to its training data and provides references to developers. It can also scan code for security issues. Its performance and correctness have shown improvement over time. Like other tools, its generated code can have security issues.

ChatGPT: Primarily known as a conversational AI, ChatGPT is also evaluated as a code generation tool that can generate code from natural language prompts. It has demonstrated capabilities in generating programs, classes, functions, and test cases from natural language descriptions. In empirical studies comparing it with GitHub Copilot and Amazon CodeWhisperer on code generation tasks, ChatGPT showed the highest success rate in generating correct code solutions when provided with clear problem descriptions, such as function signatures and docstrings [1]. A key feature noted in the study is that ChatGPT can explain its recommendations in detail, making it potentially useful for understanding the generated logic. Unlike Copilot and CodeWhisperer, it typically provides one suggestion at a time unless prompted otherwise and does not have native IDE support. It cannot access local files and consistently generates the whole function in a single interaction. It's also important to note that ChatGPT can be used for tasks other than pure code generation, like data augmentation and supporting software testing education. performance is noted to strongly correlate with the clarity and detail of the input prompt.

IV. Bridging the Gap: Benefits and Empirical Evidence

The integration of Artificial Intelligence (AI) into Low-Code/No-Code (LCNC) platforms presents significant benefits, not only by empowering non-technical users but also by augmenting the capabilities and efficiency of professional software engineers. This section highlights several ways this synergy is impacting software development practices and developer productivity.

A. Enabling Citizen Developers

AI augmentation in LCNC platforms fundamentally empowers citizen developers by making software creation more accessible and less dependent on traditional programming skills. These platforms, historically built on visual programming and drag-and-drop interfaces to simplify development, are being further enhanced by AI capabilities that abstract away complexity [3].

Specific benefits for citizen developers include:

Enhanced Discoverability and Usability: AI-powered features like Natural Language Processing (NLP) and Natural Language Interfaces (NLIs) allow users to describe desired functionalities or components in everyday language. This makes it easier for CDs to find and utilize the right tools and components within the platform without needing to know technical jargon or API structures. The simple and usable design of LCNC platforms, enhanced by AI, hides the complexity of actual coding and simplifies computational concepts, making it easy to instantly edit functionalities in real-time [4]. User-friendly no-/low-code assistance systems and AI platforms are seen as promising approaches to enable non-experts to actively participate in software development processes [7].

Ability to Tackle More Complex Tasks: With AI assistance, tasks that previously required advanced programming knowledge can become accessible to citizen developers [3]. GPT and other AI models can help users create application components based on natural language descriptions, enabling CDs to build more sophisticated applications than previously possible with LCNC alone.

Democratization of Application Development: The innovations brought by AI-assisted LCNC democratize application development. By providing business professionals with the ability to learn, develop, and employ new technical skills

with little or no coding experience, LCNC, augmented by AI, contributes to solving the shortage of traditional software developers [3]. This empowerment allows businesses to innovate faster and reduces the reliance on professional developers for all digital solutions. Studies suggest that LCNC platforms could indeed democratize software development, enabling organizations to develop necessary applications cost-effectively [2]. The democratization of AI and the associated expansion of the user base are expected to lead to new, diversified, and more innovative AI solutions [7].

B. Assisting Software Engineers

While LCNC is often associated with citizen developers, AI augmentation also provides benefits for professional software engineers, potentially redefining their roles and enhancing their productivity.

Benefits for software engineers include:

Increased Efficiency through Task Automation: AI-augmented LCNC platforms can significantly increase efficiency by automating repetitive and time-consuming development tasks [3]. This includes automation of workflows, testing, and deployment processes that were previously done manually [5]. AI-based tools are now used across various stages of the Software Development Lifecycle (SDLC), from requirements analysis to testing and documentation, leading to considerable time savings and increased productivity [3]. Empirical evidence suggests time-based efficiency improvements, such as a study where a low-code platform reduced the time to create a module generator by 20%, or instances where development time was drastically reduced in specific use cases [4]. Efficiency benefits can manifest in terms of time, effort, collaboration, and financial cost reductions.

Easier Access and Utilization of APIs: AI assistance can make it easier for developers, including SEs, to access and use large APIs. By providing contextually-relevant suggestions and code snippets, similar to advanced code completion tools, AI reduces the need to constantly refer to external documentation, streamlining the integration of services and functionalities [1].

Focus on Higher-Value Tasks: As AI and citizen developers handle simpler application development tasks, professional software engineers can potentially focus on higher-value activities such as complex architecture design, governance, security implementation, and solving unique, challenging technical problems that require deep expertise [4]. This strategic response to the shortage of traditional software developers allows organizations to leverage their SEs' skills more effectively. By empowering employees (CDs) and reducing over-dependence on traditional developers, AI-augmented LCNC can help bridge the gap between business needs and IT capacity [3].

C. Empirical Evidence from User Studies

Empirical studies have begun to evaluate the impact and effectiveness of AI-augmented development tools, including LCNC platforms, offering insights into user performance, experience, and the realization of the purported benefits.

Key findings from these studies include:

Performance Comparisons (CDs vs. SEs): A study comparing the performance of programmers and citizen developers using a prototype no-code builder found no statistically significant difference in task correctness or processing time between the two groups for simple applications [2]. While some studies highlight perceived advantages of programming experience or report performance differences, the findings from this specific study suggest that concerns about the extensive training needed for CDs might be unwarranted for certain tasks when using such tools [2].

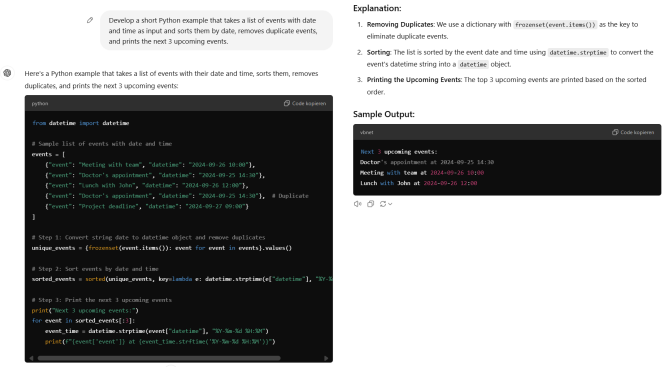

Fig. 3. This example demonstrates ChatGPT solving a basic problem by creating a well-commented Python event management script with explanations and sample output. [7].

Code Generation Tool Effectiveness: Studies evaluating specific AI code generation tools like GitHub Copilot, Amazon CodeWhisperer, and ChatGPT have compared their ability to produce correct code. One study found that ChatGPT showed the highest success rate in generating correct code solutions when given clear problem descriptions [1]. Tools like Copilot and CodeWhisperer integrate directly into IDEs and provide multiple suggestions, while ChatGPT typically offers one suggestion and can explain its output in detail. Developers use tools like Copilot to accelerate known tasks and explore new ideas.

Efficiency Metrics: Beyond correctness, studies measure efficiency. As noted earlier, empirical results indicate time savings in specific LCNC/low-code development scenarios. Effort-based efficiency, related to reducing developer hours, is also a recognized benefit [4].

User Experience and Trust: Research indicates developers use AI assistants frequently despite security concerns [4]. However, studies highlight that inexperienced users may tend to trust AI-generated code blindly, which can lead to safety risks [7]. Experienced developers are better at assessing security and improving code quality by adjusting input, thereby increasing their productivity. The need for humans to check AI output is crucial, although the process for doing so effectively is still being researched [5]. AI-assisted development tools are considered relevant to competitiveness and can reduce personnel costs.

V. Challenges and Limitations

Despite the significant potential of AI-augmented LCNC platforms to bridge the gap between citizen developers and software engineers and accelerate digital transformation, their adoption and effective utilization are accompanied by a range of challenges and limitations. These challenges span user capabilities, technical complexities, and organizational dynamics, requiring careful consideration for successful implementation.

A. User and Skill-Related Challenges

While AI aims to make development more accessible, user-related challenges persist, particularly concerning skill gaps and the nature of AI assistance itself.

A significant challenge is that AI-augmented tools still struggle to support novices when they lack clarity on their goals. Natural language interfaces and other AI aids are most effective when users can clearly articulate what they want to achieve. When users are unsure about their objectives or how to break down a complex task, current AI tools may provide less helpful or even confusing suggestions [7].

There is also a current lack of clarity on the specific ”digital skills” required for effective citizen development using LCNC platforms [4]. While LCNC reduces the need for traditional coding, effective use of these platforms, especially when augmented by AI, may require different skills such as computational thinking, creativity, and the ability to understand user experience, data mastery, and potentially prompt engineering. This lack of a clear definition contributes to concerns about the potential fear of extensive training needs. Studies suggest that while some training periods might be less extensive than feared for certain simple tasks, the necessary skills for more complex use cases remain an area needing further definition [2].

Furthermore, there can be potential interaction problems based on user knowledge and system requirements [4]. A user's existing knowledge and their understanding of the system's requirements can influence how effectively they interact with the AI-augmented LCNC platform and its features.

B. Technical and Platform Challenges

The technological foundation and integration of AI into LCNC platforms introduce their own set of hurdles.

A notable challenge is the fragmentation of LCNC development platforms [4]. There are numerous platforms available, each with different features and capabilities, which can impact their suitability for different software development tasks. This fragmentation can make it difficult for organizations to choose the right platform and may lead to inconsistent development requirements. There are inherent challenges in integrating AI capabilities, including ethical, security, and scalability concerns.

Security risks are particularly prominent [3]. AI-generated code in LCNC platforms can potentially contain vulnerabilities, and inexperienced users may not have the expertise to assess the security of this code. Issues like shadow IT, where citizen developers create applications without IT oversight, can exacerbate data privacy and security concerns [4].

Scalability can also be a challenge. While LCNC platforms can support large-scale applications, utilizing them for large-scale computations and cloud resources may require advanced knowledge and can be expensive [4]. The inherent limitations of LLMs, such as context length, can also pose challenges in handling extensive codebases within AI-assisted environments [8].

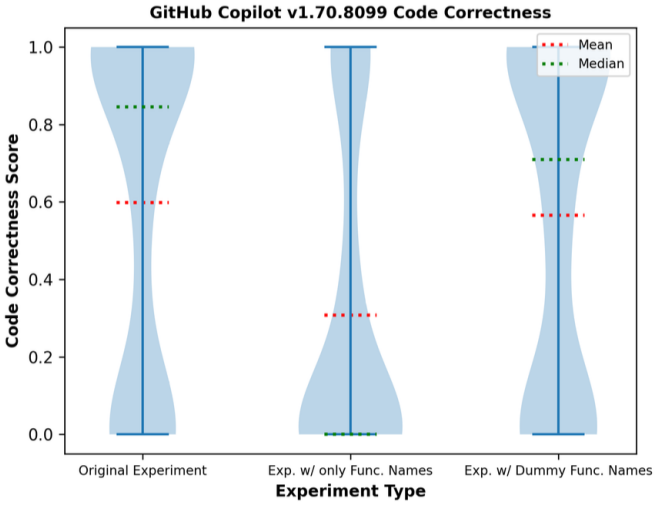

Fig. 4. Distribution of Code Accuracy Scores Across Problems in Various Experiments [1].

Moreover, the concept that LCNC can sometimes be ”high-code” means that while the initial development uses visual interfaces, users (potentially including SEs involved in governance or complex integrations) may still need to read and modify generated code. This highlights the general need for humans to check AI output, as AI models can hallucinate or produce incorrect/nonsensical code [8]. This necessity, especially for novice users who might trust AI blindly, can potentially lead to less understood code and technical debt if not properly managed [1].

C. Organizational and Collaborative Challenges

Integrating AI-augmented LCNC into existing organizational structures and fostering collaboration between different developer groups presents additional challenges.

Potential issues include role conflicts between citizen developers and professional software engineers. As CDs become empowered to develop applications with less IT intervention, SEs may perceive a loss of control or influence, affecting collaboration. Disputes over security or adherence to standards for CD-developed applications can further exacerbate these conflicts [4].

There is a recognized need for clear performance metrics for CD success. Evaluating the effectiveness and impact of citizen development initiatives within an organization, particularly when AI is involved, lacks standardized and consistent metrics [4]. Existing studies often rely on qualitative measures, and

there is an urgent need for more empirical studies on how CD performance is measured in practice.

Finally, there are significant challenges related to governing development across different user groups and ensuring quality and maintainability in an AI-augmented LCNC environment. Establishing clear rules, standards, and guidelines for citizen developers is crucial for effective IT governance and can help prevent issues like shadow IT and system integration problems. Ensuring the quality and long-term maintainability of applications developed by both CDs and SEs using AI-augmented LCNC platforms requires robust governance frameworks and potentially revised development strategies [4].

By addressing these multifaceted challenges, organizations can better navigate the complexities of adopting AI-augmented LCNC and maximize its potential benefits while mitigating associated risks.

VI. Research Gaps and Future Directions

The integration of AI into Low-Code/No-Code platforms is a relatively new phenomenon, and our review highlights several areas requiring further research to fully understand and harness its potential while addressing associated challenges. Based on the current state of the field, a clear agenda for future research emerges, focusing on refining AI assistance, clarifying skill requirements, improving platforms, understanding user-technology interaction, and addressing organizational complexities.

A. Improving AI Support for Novices

A significant challenge identified is that AI-augmented tools still struggle to support novices, particularly when they lack clarity on their goals [8]. While natural language interfaces (NLIs) aim to make interactions intuitive, users who are unsure about what they want to build or how to articulate complex requirements may find current AI suggestions less helpful or even confusing. Therefore, future research should focus on developing AI assistants that can better guide users who are unsure about their task objectives. This could involve AI systems that can engage in clarifying dialogues, help users break down vague goals into actionable steps, or offer more contextually relevant suggestions based on limited input [6]. Research is needed to explore effective AI strategies for understanding user intent when it is poorly defined and providing supportive guidance without overwhelming or distracting novice users.

B. Defining Digital Skills and Training

The widespread adoption of citizen development using LCNC platforms, especially when augmented by AI, requires a new understanding of the necessary skills. Currently, there is a lack of clarity on the specific ”digital skills” required for effective citizen development using LCNC [4]. While LCNC reduces traditional coding needs, users may require skills like computational thinking, creativity, data mastery, and an understanding of user experience. This ambiguity contributes to concerns about the potential fear of extensive training needs. Future research needs to clarify the specific digital skills required for LCNC and CD [2]. Studies could empirically investigate which skills are most critical for success in different LCNC development scenarios and with varying levels of AI assistance. Based on a clearer definition of these skills, research is also needed to develop effective training methodologies tailored to citizen developers, focusing on building these specific digital competencies.

C. Harnessing AI for Platform Design

Beyond assisting users, there's potential for AI to play a role in the design and evolution of LCNC platforms themselves. Research should advocate for exploring how AI can be used to design and optimize LCNC platforms themselves to enhance accessibility, usability, and support for multi-dimensional applications [4]. This could involve AI analyzing user interaction data to identify pain points or inefficiencies in the platform interface and suggesting design improvements. AI might also be used to automatically generate or adapt user interface components and templates to better suit specific development tasks or user needs, making platforms more flexible and accessible [7].

D. Task-Technology Fit in AI-Augmented LCNC

The fragmentation of LCNC development platforms, each with varying features and capabilities, impacts their suitability for different software development tasks. Different platforms may have similar or dissimilar characteristics that determine their fit for addressing specific needs [4]. This raises questions about how AI augmentation interacts with platform features and task requirements. Future studies should examine how specific AI-augmented features of LCNC platforms align with different software development tasks and impact user performance. Research on task-technology fit in this context could provide valuable insights for organizations choosing platforms and for developers designing applications, ensuring that the chosen AI-augmented platform is well-suited to the intended use cases and user profiles. Studies could compare user performance, efficiency, and satisfaction across different platforms and with varying levels of AI assistance for diverse tasks.

E. Evaluating Collaboration and Governance

The introduction of AI-augmented LCNC affects the dynamics between citizen developers and professional software engineers and necessitates robust organizational strategies. Potential issues such as role conflicts may arise, and there is a recognized need for clear performance metrics for CD success. Research is needed to evaluate the long-term impacts of AI-augmented LCNC on collaboration dynamics between CDs and SEs. This includes investigating how AI assistance influences communication, knowledge sharing, and perceived roles. Furthermore, research should explore the organizational strategies needed for effective adoption, including governance, roles, skills, and performance evaluation. This could involve studying how organizations can establish effective governance

frameworks to manage development across different user groups, ensure application quality and maintainability, and develop appropriate metrics to measure the success and impact of AI-augmented citizen development initiatives.

F. Proactive AI Assistance in LCNC

Current AI assistance often requires explicit user interaction, such as formulating prompts. A future direction involves exploring proactive AI agents within LCNC interfaces to offer timely assistance without disrupting user workflow. This could involve AI monitoring user actions and context within the LCNC environment and proactively suggesting relevant components, detecting potential errors before they occur, or offering guidance on best practices. Research is needed to determine how to implement such proactive assistance effectively without being intrusive or distracting, ensuring that the AI seamlessly integrates into the user's thought process and workflow. This aligns with the broader trend towards developing more sophisticated LLM-based agents for software engineering tasks that can handle complex workflows and make intelligent decisions.

VII. Conclusion

This literature review has systematically examined the convergence of Artificial Intelligence (AI) and Low-Code/No-Code (LCNC) development, focusing on its potential to bridge the gap between citizen developers (CDs) and professional software engineers (SEs). The increasing demand for software solutions necessitates approaches that extend development capabilities beyond traditional programming. LCNC platforms have emerged as a key technology to democratize software creation, making it accessible to non-technical users. Concurrently, AI has become increasingly integrated into software engineering practices to assist professional developers. Our review highlights that the integration of AI capabilities within LCNC platforms represents a novel opportunity to potentially reduce the technical divide between CDs and SEs and enhance collaboration.

The synthesis of the literature reveals several key benefits associated with AI-augmented LCNC development. These platforms, powered by AI features such as Natural Language Interfaces (NLIs), intelligent search, and code generation, empower citizen developers by making the development process more intuitive and accessible. Specifically, AI can enhance the discoverability of components and functionalities, allowing users to tackle tasks that previously required advanced programming skills. This democratization of application development enables businesses to innovate faster and reduce reliance on professional developers for simpler solutions. For software engineers, AI-augmented LCNC can increase efficiency by automating repetitive tasks and providing easier access to large APIs. This allows SEs to potentially focus on more complex or strategic aspects like architecture and governance. Empirical evidence from user studies suggests improvements in discoverability and iterative composition in AI-augmented LCNC environments.

However, the adoption of AI-augmented LCNC also presents significant challenges and limitations that require careful consideration and further research. A notable challenge is that AI-augmented tools still struggle to effectively support novices, particularly when users are unclear about their task objectives. Furthermore, there is a recognized lack of clarity regarding the specific "digital skills" required for effective citizen development using LCNC, leading to potential concerns about extensive training needs. Technical challenges include the fragmentation of LCNC platforms, where varying features impact their suitability for different tasks, and inherent issues related to AI integration such as security and scalability. Organizational and collaborative challenges include the potential for role conflicts between CDs and SEs and the need for clear performance metrics to evaluate citizen development success. The review also indicates that while AI can generate code, human oversight and verification are still necessary, potentially leading to less understood code bases.

In conclusion, while AI-augmented LCNC development holds significant promise for expanding software creation capabilities and fostering collaboration between diverse developer profiles, it is a rapidly evolving domain with many open questions. Fully realizing the synergistic potential of AI and LCNC for diverse developer communities requires continued research across various dimensions, including refining AI assistance for novice users, clarifying skill requirements and training methodologies, exploring AI's role in platform design, understanding task-technology fit, and addressing organizational and collaborative dynamics.

References

[1] B. Yetiştiren, I. Özsoy, M. Ayerdem, and E. Tüzün, "Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt," 2023. [Online]. Available: https://arxiv.org/abs/2304.10778

[2] T. Guthardt, J. Kosiol, and O. Hohlfeld, "Low-code vs. the developer: An empirical study on the developer experience and efficiency of a no-code platform," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS Companion '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 856–865. [Online]. Available: https://doi.org/10.1145/3652620.3688332

[3] G. Kacheru, N. Arthan, and R. Bajjuru, "Artificial intelligence (ai) for low-code and no-code development: Making non-developers developers in 2024," *Formosa Journal of Multidisciplinary Research*, vol. 4, no. 1, p. 141–150, Jan. 2025. [Online]. Available: https://journal.formosapublisher.org/index.php/fjmr/article/view/13369

[4] M. O. Ajimati, N. Carroll, and M. Maher, "Adoption of low-code and no-code development: A systematic literature review and future research agenda," *Journal of Systems and Software*, vol. 222, p. 112300, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121224003443

[5] O. Bruhin, E. Dickhaut, E. Elshan, and M. Li, "The rise of generative ai in low code development platforms – an analysis and future directions," Jan 2024. [Online]. Available: https://www.alexandria.unisg.ch/handle/20.500.14171/118148

[6] S. Joel, J. J. Wu, and F. H. Fard, "A survey on llm-based code generation for low-resource and domain-specific programming languages," 2024. [Online]. Available: https://arxiv.org/abs/2410.03981

[7] S. Torka and S. Albayrak, "Optimizing ai-assisted code generation," 2024. [Online]. Available: https://arxiv.org/abs/2412.10953

[8] H. Jin, L. Huang, H. Cai, J. Yan, B. Li, and H. Chen, "From llms to llm-based agents for software engineering: A survey of current, challenges and future," 2025. [Online]. Available: https://arxiv.org/abs/2408.02479