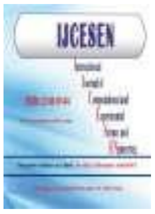
International Journal of Computational and Experimental Science and ENgineering
(IJCESEN)

Vol. 11-No.3 (2025) pp. 5559-5565
http://www.ijcesen.com

Copyright © IJCESEN



ISSN: 2149-9144

Research Article

Large Language Model Framework for Device Orchestration in Low-Code No-Code Solutions

Rohith Kumar Punithavel[1*], Deeksha Sivakumer[2]

[1]Robert W. Plaster Graduate School of Business, University of the Cumberlands, Kentucky, USA
*** Corresponding author Email:** rohithkumar.punithavel@gmail.com - **ORCID:** 0009-0002-3922-5549

[2]Robert W. Plaster Graduate School of Business, University of the Cumberlands, Kentucky, USA
Email: sivakumer.deeksha@gmail.com - **ORCID:** 0009-0002-3922-5548

Article Info:

DOI: 10.22399/ijcesen.3521
Received: 21 May 2025
Accepted: 18 July 2025

Keywords

LLM,
Prompt Engineering,
Low-Code
No-Code

Abstract:

In today's digital era, virtually every aspect of life or industry is influenced by digital services. Digital devices have played a critical role in transforming various fields, from education to healthcare, by enhancing availability and access. They have also helped prevent theft, robbery, and even property damage. They have also contributed to crime prevention, including theft, robbery, and property damage. With exponential growth, digital device manufacturers serving similar or different purposes have created a heterogeneous digital ecosystem to offer ease-of-use solutions. However, this heterogeneity poses challenges even for software developers. Beyond software developers, a growing number of professionals from other engineering streams, management, and small to medium-sized business owners are required to interact with digital systems. Low-code and no-code platforms have emerged as viable solutions to support this shift and encourage these users to manage digital services without complex programming. The Low-Code and No-Code platforms have limitations but are not limited to flexibility, usability, and integration, which restrict the development of customized solutions. Large Language Models have recently drawn everyone's attention due to their cognitive ability and capability to bridge gaps. In this paper, we explore how LLMs can enhance Low-Code and No-Code platforms by facilitating the integration of digital devices and improving the management of their usage and services.

1. Introduction

The Internet of Things (IoT) has existed for several decades. IoT is simply an interconnected network of devices that allows communication among and between them and the cloud. Through advancements in connectivity, IoT has become an everyday feature of life, from home appliances to streetlights. IoT devices usually consist of sensors, actuators, and communications interfaces, allowing them to form a global network that can exchange data over the Internet without significant human involvement. IoT transforms devices into intelligent entities and emphasizes the creation of an interconnected ecosystem where cloud computing, RFID, and Wireless sensor networks work together to ensure data integrity and security (Gubbi et al., 2013). The applications of IoT span across various domains, including military, healthcare, home automation, solid waste management, surveillance systems, consumer asset tracking, smart grids, vehicular communication systems, agriculture, transportation, and disaster monitoring. In the military, IoT is used for health monitoring, tracking of armed personnel, smart uniforms, situational awareness, and naval monitoring. IoT supports remote health monitoring, ingestible sensors, mobile health solutions, and smart hospitals. Home automation is enhanced through remote control of appliances, smart lighting, intrusion detection, and elder care systems. IoT has enabled solutions centered on sensor-based monitoring, like water leak detection, real-time and remote tracking of utilities usage like water, electricity, and gas, and many more. In security, IoT systems can monitor activities from remote locations through audio and video, incorporating motion and face detection, and providing alerts in real-time. In connected assets, IoT systems integrate GPS, RFID, and accelerometers; in e-commerce, IoT has provided the ability to trace and track within logistics. In agriculture, transportation, and disaster management, IoT systems are providing solutions around crop management, nutrient monitoring, road traffic control, parking management, toll collection, fleet, and disaster management by helping in providing early warning of tsunamis, forest fires, floods, and landslides (Ramson et al., 2020). As the demand for the IoT has evolved to where we are today, many vendors and solution providers have risen and are selling their solutions, resulting in a fragmented ecosystem. This fragmentation has made integrating and testing IoT systems challenging and hindered seamless interoperability (Dudhe et al., 2017).

Low-Code and No-Code platforms allow users with little to no coding experience to create software applications. Software Development has been drastically changed through Low-Code and No-Code platforms, which simplify the development approach and provide a faster, easier, and less expensive alternative to traditional software development. The Low-Code and No-Code platform has drag-and-drop components, visual tools, and prebuilt modules, which can decrease development time and complexity. The Low-Code and No-Code platform has easy-to-use interfaces, automatic security updates, agility, and less maintenance (Shridhar, 2021). One of the slightly lesser challenges is that Low-Code and No-Code platforms have limitations based on their capabilities or available components, meaning that Low-Code and No-Code are not the best approach for developing functionality that requires high customization or complex functionality (Elshan et al., 2023). However, one unfortunate challenge of Low-Code and No-Code is integrating heterogeneous ecosystems (e.g., Internet of Things (IoT)). Heterogeneous ecosystems create multiple obstacles to seamless interoperability, scalability, and maintainability. The implications of limits on interoperability can affect the broader uptake and use of Low-Code and No-Code platforms.

A Large Language Model (LLM) is an artificial intelligence system designed to understand and generate human-like text. LLMs are based on artificial neural networks with a transformer architecture. The architecture can process language efficiently and at scale (Naveed et al., 2020). LLMs predict the next word in a sentence through a training process that involves learning statistical relationships between words from large amounts of data. They do not possess understanding, belief, or consciousness; instead, they generate responses based on learned patterns, not reasoning or experience (Shanahan, 2024). Over the last decade, LLMs have evolved significantly with the introduction of transformer architecture. The transformer architecture uses an attention mechanism to understand sentence relationships simultaneously, allowing LLMs to perform tasks with minimal instruction. GPT-2 and GPT-3 show that LLMs can do a lot of different things with little or no instruction when they have more data and parameters (Naveed et al., 2020). Modern models like GPT-4 are multimodal, capable of understanding text, images, and audio. Modern LLMs are further fine-tuned with techniques like Instruction Tuning and Reinforcement Learning from Human Feedback to enhance their accuracy and performance in complex scenarios (Teubner et al., 2023). Large Language Models (LLMs) offer advantages like zero-shot and few-shot generalization, in-context learning, multi-task transfer, and emergent abilities like reasoning and planning due to their scale and transformer architecture. LLMs suffer from hallucinations, sensitivity to prompts, and failing to align context, which produces biased or misleading output content. LLMs require vast computing power, capital, and energy resources to train and generate predictions, which raises doubts when considering the range of scaling. Even with human-like physical attributes that could provide the observer a sense of interaction, LLMs demonstrate no understanding or reasoning and operate by next-token prediction of statistical patterns in training data.

LLMs are used in various language tasks like text summarization, machine translation, information retrieval, question answering, and dialogue generation. They support code generation and completion in programming environments. LLMs are utilized in multimodal settings for tasks like image captioning, audio-text reasoning, conversational agents, assistive writing tools, autonomous systems, and educational tutors. LLMs are also embedded in business and decision-support systems to enhance productivity and content generation (Naveed et al., 2020; Shanahan, 2024; Teubner et al., 2023). Large Language models (LLMs), particularly large-scale transformer-based models, require substantial computational infrastructure for training. This includes large datasets, high-performance parallelized hardware, distributed training strategies across multiple devices, mixed-precision training to reduce memory and computational requirements, and memory and compute efficient techniques to enable model scaling and performance. High operational costs due to intensive computing per query contribute to financial burdens. Therefore, research is needed to develop more efficient architectures, tuning methods, and inference optimizations for LLMs to be more scalable and sustainable (Teubner et al., 2023).

This paper presents a framework that addresses the challenge of onboarding heterogeneous IoT devices on low-code and no-code platforms using large language models. In Section 2, we survey existing solutions that address IoT Interoperability issues, Low-Code and No-Code platform solutions, and applications of LLMs. Section 3 discusses the proposed end-to-end architecture of our proposed methodology. Finally, we summarize the framework's key findings, limitations, and future scope.

2. Literature Review

The paper by Deshmukh et al. (2021) introduces a middleware product called 'Data Spine' and officially aims to solve the very real issue of interoperability of heterogeneous IoT platforms. The Data Spine product provides a federation of interconnected services and platforms, enabling, in theory, on-demand interoperability by leveraging adaptive data mapping and transformation functionalities. Deshmukh et al.'s (2021) research contributes significantly to the field of IoT by providing a complete architectural framework and implementation strategy for enabling cross-platform communication. The innovative aspect of the Data Spine product allows integration via a visual, drag-and-drop style interface, allowing users to bridge the gap introduced by the protocol and data model contradictions, while having limited technical overhead. By utilizing the standardized API descriptions, the Data Spine limits the effort required for onboarding new platforms and guarantees the consistency and discoverability of services. In addition, the paper describes how this system supports scalable integration by providing an extensive library of built-in transformation, routing, and mediation components to address performance, availability, and maintainability challenges found in innovative manufacturing spaces. Despite the Data Spine taking an excellent first step towards integrating heterogeneous IoT services, the approach relies heavily on the manual configuration of its foundries and human interpretation of the API documentation.

The paper by Liu et al. (2024) performs a comparative analysis of 1600 discussions from Stack Overflow on traditional low-code programming (LCP) and LCP augmented by large language models (LLMs). The Liu et al. (2024) research highlights challenges of traditional low-code programming (LCP) and LCP with LLMs. Traditional LCP requires professional programming knowledge for complex functionality beyond drag-and-drop interfaces. In most cases, a user will need programming knowledge to understand the necessary code to implement complex functionality or to troubleshoot or customize its behavior. Integrating with new APIs is not possible because of the platform's design. Along with this reliability, version mismatch and compatibility are some of the most common issues in Traditional LCP. Regarding generalized LLM-based LCPs, reliability and trustworthiness continue to be problems. Generalized LLMs produce errors due to outdated training data, and users will need programming knowledge to validate or correct the outputs. LLMs are also often prone to hallucinations, where they produce functions or libraries that do not exist. Privacy concerns arise as users worry about exposing sensitive data when interacting with proprietary LLMs during code generation. The proposed framework is a structured, secure, and semantically aware approach to low-code automation that has been specifically designed to integrate with protected internal resources and will use a custom-trained LLM, which will parse technical documentation and create sound, domain-specific code blocks.

Hagel et al. (2024) proposed a framework for the conversion of Low-Code Development Platforms (LCDPs) to a natively No-Code platform via Large Language Models (LLMs). The LLM framework allows users to enter specifications using natural language, which large language models translate into valid domain-specific models. This research paper established a prompt template to establish the domain-specific grammar and an example data model to enhance the syntactic and semantic accuracy during processing. The paper by Hagel et al. (2024) compared traditional modeling against LLM-assisted modeling and showed a significant reduction in task completion time without negatively affecting usability. The paper by Hagel et al. (2024) aligns with our research's core objective of using LLMs to bridge the gap between user intent and application generation within Low-Code and No-Code Platforms. The Hagel et al. (2024) study's intention to use prompt engineering to translate input as domain-specific models related directly to our research intention of onboarding heterogeneous devices into a single platform. Hagel et al. (2024) assume the user will provide a clear, structured specification for No-Code development. The method also lacks mechanisms to interpret formal API schemas, metadata, or protocol details. It is based on a single, predefined DSL for form generation, which limits its generalizability to heterogeneous domains like IoT.

Wang et al. (2025) examines the possible use of large language models for Low-Code and No-Code development of end-user IoT applications. The paper offers a detailed evaluation framework to assess LLMs' performance in this area of interest. The authors expanded a dataset of user-described smart home tasks by adding multilingual prompts and detailed semantic evaluation metrics. They used the LLM4FaaS platform, which combines LLMs with Function-as-a-Service infrastructure, aggregating the importance of LLMs to generate only core functional logic. Wang et al. (2025) examined the effect of the linguistic provenance of the LLMs on the outputs produced, noting they achieved alignment between model-trained data and input language, enabling usability improvements and appropriateness for an end-user audience. The authors noted issues for domain-specific functionality and lightweight models. They outlined further reflections for LLM deployment in applications related to usability, user trust, and ethical considerations. Ultimately, the study underscores the importance of model selection, prompt design, and linguistic alignment in applying LLMs for no-code IoT development. Wang et al. (2025) is focused on building automation rules and workflows with descriptive prompts, leveraging LLMs to overcome the coding barrier to customizing application behavior. Their work is centered around the end-user, taking inputs from users. Our proposed work addresses a related problem and is focused on onboarding heterogeneous devices and access devices capabilities through low-code and no-code platforms, leveraging large language models. Our framework is built around the interoperability of devices through a single schema, which can abstract away vendor-specific differences, moving from user-driven scripted behavior to structured-device onboarding.

3. Proposed Framework

The proposed framework integrates three important components: devices, Low Code No Code User Interface, and Large Language Model.

Devices

The devices under consideration are heterogeneous. They differ in data formats, processing power, communication protocols, hardware systems, and capabilities. APIs are used by these devices to facilitate smooth integration and interaction. The APIs not only offer remote control and automation but have broadened the device's capabilities, allowing it to perform operations like data collection, device management, and device monitoring.

Low Code No Code platform

Low-Code and No-Code platforms comprise independent components that support quick application development through abstractions, minimal coding, and seamless integration. The Low-Code No-Code platforms are organized into layered structures starting with an application modeler that provides a graphical interface for users to interact with, like drag and drop tools, templates, etc. Next is server-side infrastructure that handles backend functions like code generation, runtime model interpretations, microservices orchestration, databases, and APIs. The third layer contains external services interfaced through standard protocols like third-party data connectors, cloud services, analytical tools, etc. The key considerations while designing Low-Code No-Code platforms are the ability to work with diverse external systems, scale to accommodate varying traffic loads, and security. The platforms also prioritize collaborative development, reusability, and flexibility (Sahay et al., 2020).

Large Language Model

The proposed framework utilizes the Large Language Model as its core component, with detailed implementation details provided in three sections: Model Selection, Data Preparation, and Tuning.

Model Selection

LLMs have specific strengths and tradeoffs, and choosing the right one depends on alignment with your workload. Comprehending the performance of various models across fundamental tasks aids in developing more intelligent, rapid, and effective systems. With more models emerging each year, choosing a model based off of its capability is important. In this paper, we explore how to select a model and fine tune for the task by laying down three important research questions. RQ1: How accurate and semantically correct is the generated code for the use case? RQ2: Is the generated code efficient and optimized? RQ3: How well does the LLM understand, interpret, and perform to diverse input prompts from non-technical users?

The first research question measures a model's capability to rewrite user text prompts into syntactically valid and semantically valid code. This question is fundamental in a domain-specific context, such as onboarding IoT devices. The semantics of user text prompts to executable code are critical, especially for an end user of the system, who may have little knowledge of how to code. There is also a challenge in applying their intent to a range of potentially complex and diverse API specifications of IoT devices. The LLM needs to consistently generate accurate code that compiles and operates as intended by the user (Wang et al., 2025). The second research question concerns performance efficiency, which is crucial for latency-sensitive or resource-constrained deployments. The metrics like execution time, memory usage, and performance variance compared to human code are central to benchmark studies. LLMs generally create functionally correct code blocks, but much variation exists in runtime performance. Coignion et al. (2024) researched 18 LLMs with a Leetcode dataset, measuring and analyzing the model's runtime and memory performance against human code. Finally, they found that the LLMs produced more efficient code than humans, but models like CodeLlama and StarCoder perform well in optimizing algorithms. Models like CodeLlama and StarCoder perform well in optimizing algorithms, while others show higher resource consumption or failure under large input loads (Coignion et al., 2024). The third question addresses prompt handling, which is essential to the usability of LLM-based approaches. In this proposed framework, non-technical users depend on natural language to explain their requirements for visualization, module development goals, upload API information, request API customizations, or define metadata such as an endpoint URL or access server information. As a result, the LLM needs to interpret various prompts, many vague and imprecise, and support interactive, multi-turn conversations. Wang et al.'s (2025) research shows that reasoning-optimized models like Gemini and DeepSeek R1 consistently displayed higher prompt robustness and consistent context-management during step-by-step configurations compared to others. They effectively used chain-of-thought prompting to allow the user to process through complicated workflows without burdening them with excessive technical detail.

Combining the findings of Coignion et al. (2024), Wang et al. (2025), and Joel et al. (2024) the three best LLMs for this use case are: GPT-4o, StarCoder and Gemini. GPT-4o was the strongest on accuracy, adaptability and reasoning. StarCoder, because of its high-performance code generation, and Gemini, because of its prompt interpretation and human relation.

Dataset

To build a high-quality dataset for a custom LLM that can onboard API-enabled devices via a chatbot interface, a systematic multi-step strategy that combines existing resources, structured transformation, and synthetic augmentation is required. The readily available datasets for API information include OpenAPI specifications from APIs.guru and OpenAPI Directory and Postman Collections scattered on GitHub. To extract structured metadata, including endpoints, methods, parameters, and authentication schemes, these datasets can be parsed programmatically using tools like Swagger-parser or Postman SDKs. The coding datasets, such as HumanEval and MBPP, can be very useful in creating a dataset that can augment the LLM in a basic code synthesis and logic composition way. Also, Leetcode datasets can help provide a performance and execution environment benchmark towards ensuring the production of executable code and data usages in a performance-sensitive and resource-constrained environment (Coignion et al., 2024). The API and coding data can now be transformed into simulated user interactions, mimicking a step-by-step onboarding dialogue. Joel et al.'s (2024) research on LLM applications for both domain-specific and low-resource programming languages suggests that dataset augmentation using synthetic dialog generation, chain-of-thought prompting, and scenario-based fine-tuning will be essential in compensating for a lack of data and keeping the training consistent with actual workflows. The dataset should include the user prompt, API document, LLM's reasoning chain, code modules, and validation tests. Such a hybrid approach should allow the LLM to leverage options to appropriately parse the documentation and generate executable modules for the Low-Code No-Code interface.

Fine Tuning

Training LLMs from scratch consumes a lot of time and capital costs. Fine-tuning is taking an already trained Large Language Model (LLM) and continuing the training on a smaller, specifically tasked dataset to adjust the model for a specific purpose, style, or application. Raj et al.'s (2024) research addresses the significance of fine-tuning methods for Large Language Models (LLMs) in terms of data, memory optimization methods, and focused training strategies. A few techniques include Low-Rank Adaptation (LoRA) and Quantized Low-Rank Adaptation (QLoRA) that require training only a few small, trainable matrices used within the model architecture, reducing the number of trainable parameters. LoRA allows models to learn when task adaptation is desirable with limited computational overhead, while QLoRA maintains model quality using 4-bit quantization for reduced memory use. Quantization is converting model weights from high-precision to lower-precision formats, decreasing the memory footprint of the model and energy usage during training and inference. Post-training quantization and quantization-aware training (QAT) are examined to mitigate model accuracy degradation by leveraging a training process that accounts for quantization errors. Finally, gradient accumulation was suggested to manage large batch sizes based on limited hardware while indirectly taking advantage of larger batch sizes by capturing gradients over several iterations and applying each set of computed gradients jointly, since this maintains the behavior of larger batch training when limited to hardware maximum memory capacity (Raj et al., 2024). Once the model is fine-tuned, a single evaluation metric is inadequate for evaluating the performance of a Large Language Model (LLM) in a multistage, end-to-end onboarding system. This model employs multiple targeted metrics across some key stages to provide a more reliable and comprehensive evaluation. For evaluation of prompts, possible metrics include Prompt Interpretation Accuracy or user-rated Response Helpfulness. In assessing code generation, metrics like Pass@k or Functional Correctness Rate are considered. Quantitative metrics, including BLEU, Precision, Recall, and F1-Score, can be used to evaluate a model for document parsing (Wang et al., 2025).

End-to-End Working of Proposed Framework

The proposed system integrates a custom-trained Large Language Model (LLM) within a Low-Code No-Code user interface to automate the onboarding of API-enabled devices. The user interacts with the system through a chatbot interface powered by the custom-trained LLM, designed specifically to interpret and operationalize device APIs. Vendors typically provide API information through various documentation formats, including Swagger docs, OpenAPI specifications, Postman Collections, and traditional API documentation. In the chatbot interface, the user prompts the LLM to onboard a device, LLM requests for API information document. Upon uploading the documentation, the LLM initiates a guided, step-by-step interaction with the user, employing a chain of thought prompting mechanism to collect necessary metadata such as API endpoint URLs, database configurations if data storage is required, and internal server from which any data needs to be retrieved. The LLM also requests user if any customization must be made for APIs. Following this interactive phase, the LLM autonomously generates code modules tailored to the device's API specifications. These modules undergo a validation pipeline comprising unit testing, functional testing, integration checks, and connectivity testing. Upon successful validation, the resulting modules are made available on the Low-Code No-Code platform's user interface dashboard. Users can then interact with the onboarded device through a set of autogenerated actions, such as retrieving real-time, visualizing historical data if storage was enabled, toggling device states e.g., turning devices on/off, or executing other API supported operations.

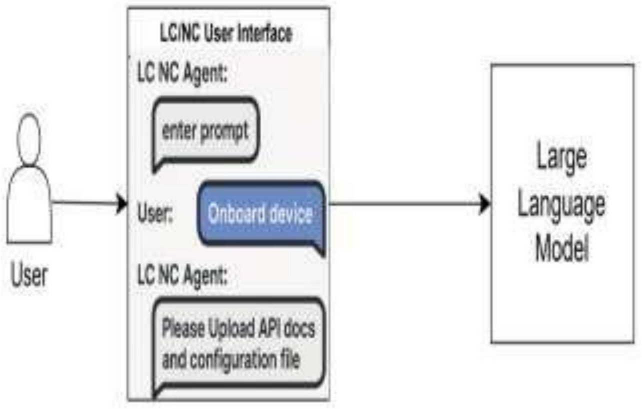


Figure 1 *Chatbot Interaction Interface of Low Code No Code Platform*

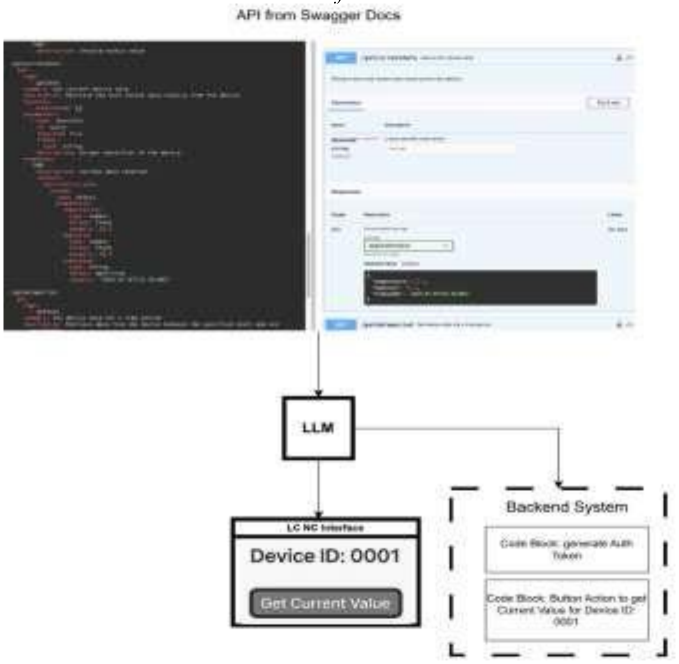

API from Swagger Docs
LLM
LC NC Interface
Device ID: 0001
Get Current Value
Backend System
Code Block: generate Auth Token
Code Block: Button Action to get Current Value for Device ID: 0001

Figure 2 *API Swagger Document (YAML file) to Code block*

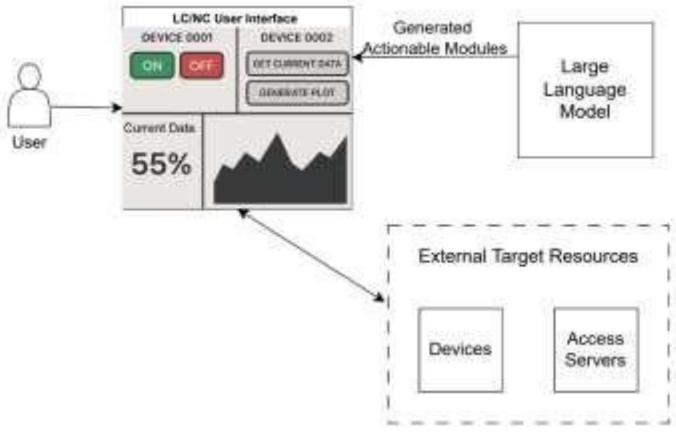


Figure 3 *Generated Actionable Modules on Low Code No Code User Interface*

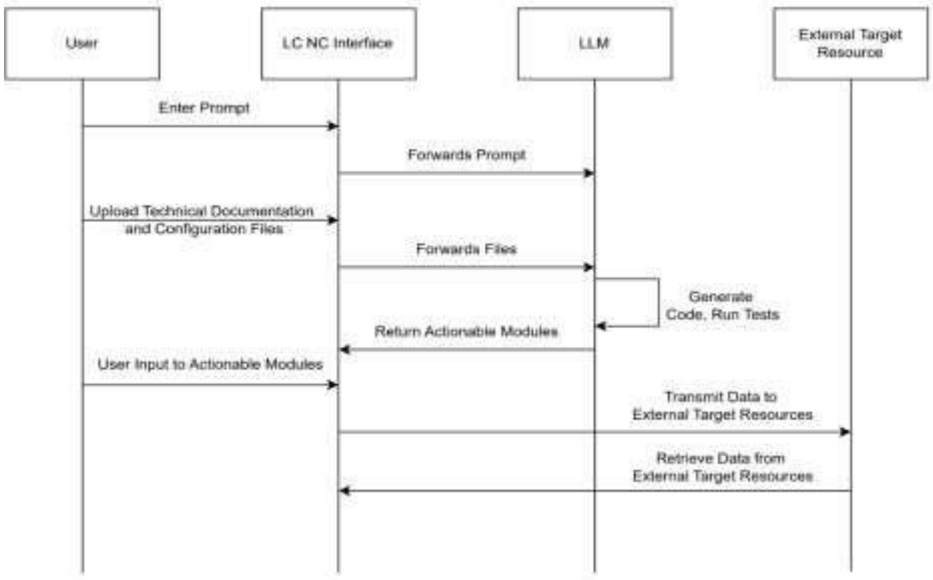

User
LC NC Interface
LLM
External Target Resource
Enter Prompt
Forwards Prompt
Upload Technical Documentation and Configuration Files
Forwards Files
Generate Code, Run Tests
Return Actionable Modules
User Input to Actionable Modules
Transmit Data to External Target Resources
Retrieve Data from External Target Resources

Figure 4 *End-to-End Orchestration Diagram*

4. Challenges

The challenges of utilizing a custom-trained Large Language Model in a Low-Code No-Code environment with heterogeneous devices are not strictly technical integration challenges. The LLM has been trained specifically for the organization's use, offering an additional layer of security and control of data privacy, as sensitive information stays in the organization's trusted domain. However, the LLM must contend with various authentication protocols each vendor has developed and provided insufficient documentation. To get the system to communicate, the LLM must identify and reason through vendor-based protocols, prompt the user to record their credential in a secure scope, and complete the task. Latency is another serious effect when using LLM, especially for live chatbots. However, sometimes using LLMs, by adding techniques such as LoRA or using quantization to reduce latency, it is possible to have some latency vs not accepting less than optimal output (Raj et al., 2024). The LLM has challenges due to a feedback loop that involves handling user corrections, interpreting testing errors, and autonomously refining code. Due to limited dataset availability, LLMs will also have difficulty onboarding devices where the vendor uses lower-level languages, like Verilog (Joel et al., 2024). The quality and consistency of generated code rely on the variability in user prompts, meaning the effectiveness of LLMs depends on non-obvious aspects of the user's natural language background and the prompt structure used. Other challenges related to onboarding using LLMs may include concerns with validation pipelines, scaling, performance, and dealing with dynamic API versioning or undocumented API changes (Wang et al., 2025).

5. Conclusion

The convergence of Low-Code No-Code platforms, Internet of Things (IoT), and Large Language Models (LLMs) presents a transformative opportunity for digital service development and deployment. This paper presents a framework that uses Language Learning Models (LLMs) to automate the integration of diverse IoT devices within Low-Code No-Code platforms, thereby reducing technical barriers that traditionally hinder device interoperability by combining domain-specific code generation with an interactive chatbot interface. Our research explores methods to tackle device ecosystem fragmentation and complexity, highlighting the limitations of current Low-Code No-Code platforms and the potential of LLMs to fill these gaps. Rapid evolution of LLM capabilities, including improved reasoning, multimodal input, and reduced latency, enhances their potential in Low-Code No-Code platforms. Future LLMs promise greater context understanding, user alignment, and domain-specific integrations like real-time device communication. The proposed framework aims to create an intelligent, adaptive, and scalable Low-Code No-Code solution for all users. However, challenges exist in protocol handling, API version drift, and compute performance. Ongoing research in tuning strategies, domain-specific datasets, and prompt engineering is vital for maximizing LLM-driven IoT integration in Low-Code No-Code ecosystems.

Author Statements:

- **Ethical approval:** The conducted research is not related to either human or animal use.
- **Conflict of interest:** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper
- **Acknowledgement:** The authors declare that they have nobody or no-company to acknowledge.
- **Author contributions:** The authors declare that they have equal right on this paper.
- **Funding information:** The authors declare that there is no funding to be acknowledged.
- **Data availability statement:** The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

References

[1] Buchmann, T., Peinl, R., & Schwägerl, F. (2024, September). White-box LLM-supported Low-code Engineering: A Vision and First Insights. In Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (pp. 556-560). https://doi.org/10.1145/3652620.3687803

[2] Coignion, T., Quinton, C., & Rouvoy, R. (2024, June). A performance study of llm-generated code on leetcode. In Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering (pp. 79-89). https://doi.org/10.1145/3661167.3661221

[3] Deshmukh, R. A., Jayakody, D., Schneider, A., & Damjanovic-Behrendt, V. (2021). Data Spine: A Federated Interoperability Enabler for Heterogeneous IoT Platform Ecosystems. Sensors, 21(12), 4010. https://doi.org/10.3390/s21124010

[4] Dudhe, P. V., Kadam, N. V., Hushangabade, R. M., & Deshmukh, M. S. (2017, August). Internet of Things (IOT): An overview and its applications. 2017 International conference on energy, communication, data analytics and soft computing (ICECDS) (pp. 2650-2653). IEEE. https://doi.org/10.1109/icecds.2017.8389935

[5] Elshan, E., Dickhaut, E., & Ebel, P. A. (2023). An investigation of why low code platforms provide answers and new challenges. 56th Hawaii International Conference on System Sciences (pp 6159) https://hdl.handle.net/10125/103380

[6] Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems (Vol. 29, pp. 1645–1660). http://dx.doi.org/10.1016/j.future.2013.01.010

[7] Hagel, N., Hili, N., & Schwab, D. (2024, September). Turning Low-Code Development Platforms into True No-Code with LLMs. In Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (pp. 876-885). https://doi.org/10.1145/3652620.3688334

[8] Joel, S., Wu, J. J., & Fard, F. H. (2024). Survey on Code Generation for Low resource and Domain Specific Programming Languages. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2410.03981

[9] Liu, Y., Chen, J., Bi, T., Grundy, J., Wang, Y., Chen, T., Tang, Y., & Zheng, Z. (2024). An Empirical Study on Low Code Programming using Traditional vs Large Language Model Support. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2402.01156

[10] Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Akhtar, N., Barnes, N., & Mian, A. (2020). A comprehensive overview of large language models. ACM Transactions on Intelligent Systems and Technology. https://doi.org/10.1145/3744746

[11] Raj, J., Kushala, V., Warrier, H., & Gupta, Y. (2024). Fine Tuning LLM for Enterprise: Practical guidelines and recommendations. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2404.10779

[12] Ramson, S. J., Vishnu, S., & Shanmugam, M. (2020). Applications of Internet of Things (IoT) – an overview. 2022 6th International Conference on Devices, Circuits and Systems (ICDCS), 92–95. https://doi.org/10.1109/icdcs48716.2020.243556

[13] Sahay, A., Indamutsa, A., Di Ruscio, D., & Pierantonio, A. (2020, August). Supporting the understanding and comparison of low-code development platforms. In 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA) (pp. 171-178). IEEE. https://doi.org/10.1109/seaa51224.2020.00036

[14] Shanahan, M. (2024). Talking about Large Language Models. Communications of the ACM, 67(2), 68–79. https://doi.org/10.1145/3624724

[15] Shridhar, S. (2021). Analysis of Low Code-No Code Development Platforms in comparison with Traditional Development Methodologies. International Journal for Research in Applied Science and Engineering Technology, 9(12), 508–513. https://doi.org/10.22214/ijraset.2021.39328

[16] Teubner, T., Flath, C. M., Weinhardt, C., Van Der Aalst, W., & Hinz, O. (2023). Welcome to the Era of ChatGPT et al. Business & Information Systems Engineering, 65(2), 95–101. https://doi.org/10.1007/s12599-023-00795-x

[17] Wang, M., Kapp, A., Schirmer, T., Pfandzelter, T., & Bermbach, D. (2025). Exploring Influence Factors on LLM Suitability for No-Code Development of End User IoT Applications. arXiv (Cornell University). https://doi.org/10.48550/arXiv.2505.04710