Using LLMs to generate mHealth applications

William Niemiec[1], **Anderson Rocha Tavares**[1], **Érika Cota**[1]

[1]Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, Brazil

{william.niemiec,artavares,erika}@inf.ufrgs.br

Abstract. *The development of mobile health (mHealth) applications is resource-intensive, and the effectiveness of Large Language Models (LLMs) in automating this process remains understudied. This paper evaluates a Multi-Agent LLM Framework (MALFRA) pipeline that translates natural-language specifications into executable code for four mHealth applications. Results show that the framework produced compilable applications with relatively few statically detected bugs and no vulnerabilities detected by the configured static-analysis rules. However, functional completeness was limited: 46.38% of the evaluated features were not implemented. MALFRA particularly struggled with complex multi-step workflows and backend-dependent functionality, indicating that successful code generation and compilation do not imply functional completeness for mHealth applications.*

1. Introduction

Mobile health (mHealth) applications have been used to improve access, efficiency, and quality of care. However, building them is still resource-intensive. This kind of application requires specialized coding skills, a large amount of time, and an extensive knowledge of the health domain [Alharbi and Alshayeb 2026]. These barriers make the creation of mHealth applications a bureaucratic process, requiring close collaboration between non-programmers ("citizen developers") and developers.

In parallel, Large Language Models (LLMs) have emerged as a powerful tool to automate software generation. LLMs contribute to software engineering by writing source code and system architecture from natural language specifications [Joel et al. 2025]. This allows developers to build software prototypes faster and with fewer technical challenges. While LLMs have been widely adopted, their particular role and effectiveness for tackling the complex challenges of mobile application development have not been extensively studied at scale.

To bridge this gap, previous research introduced a framework for the automated generation and evaluation of mobile applications, utilizing a Multi-Agent LLM Framework (MALFRA) [Niemiec et al. 2026]. While this initial investigation successfully demonstrated that LLMs could produce compilable, stable, and functional app prototypes from natural language, it was limited to a small sample of three applications. Consequently, its performance across different application domains remained unclear.

In this paper, we evaluate the MALFRA framework with mHealth applications. We conducted research involving four mHealth applications: Blink Fitness™, Blood Pressure Tracker™, MeuSusDigital™, and MyChart™. The main contributions of this paper

are twofold: (i) an empirical evaluation of LLM-based mobile application generation in the mHealth domain using four real-world applications, and (ii) an analysis of the functional limitations of the generated applications.

2. Related Work

Automatic software generation aims at reducing the effort needed to translate high-level specifications into working systems. Multi-agent conversations, along with verbal reinforcement, are managed by frameworks like AutoGen [Wu et al. 2024] and Reflexion [Shinn et al. 2023] to boost code generation tasks. These and other tools were evaluated across diverse application domains, including mathematics, retrieval-augmented question answering, and embodied decision-making.

Some researchers investigated automatic software generation in the health domain. Liu et al. describe the process of building a no-code platform for creating mHealth applications for clinical researches [Liu et al. 2022]. In a similar way, Brandon et al. propose a methodology to build low-code artificial intelligence (AI) development platforms for health informatics using model-driven development and component reuse concepts [Brandon and Margaria 2023].

While these approaches demonstrate the potential of automated software generation, limited evidence exists on generating complete mHealth applications from high-level textual specifications. Previous work [Niemiec et al. 2026] introduced MALFRA and evaluated it with three non-healthcare applications. Here, we investigate whether the same pipeline can generate applications in the mHealth domain rather than comparing alternative models or generation frameworks.

3. Proposed Approach

The proposed methodology transforms informal application specifications into executable mobile applications through the pipeline shown in Figure 1. First, an LLM converts a high-level specification into a structured application description containing UI layouts, data models, and workflow representations. MALFRA then uses this description to generate source code, which is evaluated through static analysis and feature-level functional assessment.

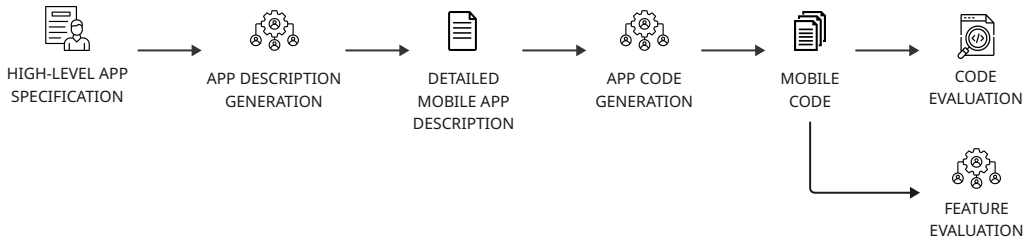


Figure 1. Pipeline used in this work.

We generated four mHealth applications: Close2BlinkFitness, Close2BloodPressureTracker, Close2MeuSusDigital, and Close2MyChart. Gemini 2.5 Pro was selected for specification generation due to its long-context reasoning capabilities, while Claude 4 Sonnet was selected for code generation due to its software-

development capabilities. Kilo Code orchestrated MALFRA, targeting Flutter[1] (Google's cross-platform UI framework based on Dart). Experiments were conducted in May 2025.

For each target application, the expected functionality was represented as a predefined set of features derived from its high-level specification and reference functionality. Each feature was manually evaluated by one software engineer with more than five years of professional software development experience. The evaluator primarily assessed features by interacting with the running generated applications and inspected the source code when necessary to clarify whether a feature had been implemented. He classified each feature as *No Issues*, *Minor Issues*, *Major Issues*, or *Not Implemented*. The expected feature set was defined before inspecting the generated applications to reduce evaluation bias.

We defined the following quantitative metrics to assess functional correctness based on expert evaluation: Feature Correctness Rate (FCR), the fraction of features with no issues; Feature Completeness (FC), the fraction with no or only minor issues; Problem Identification Rate (PIR), the fraction with minor or major issues; and Missing Feature Rate (MFR), the fraction classified as not implemented.

Internal code quality was assessed using SonarQube Community Edition 25.4.0.105899, a static-analysis platform that detects code-quality and security issues according to configurable rules. We collected detected bugs, vulnerabilities, code smells, duplication density, and cyclomatic complexity. Cyclomatic complexity measures the number of linearly independent execution paths through a function and is used here as an indicator of control-flow complexity. Security results therefore represent only issues detectable by the configured static-analysis rules and do not cover runtime behavior, logical vulnerabilities, or data leakage.

4. Results

Our results are organized in two categories: code evaluation and feature evaluation. All the experiments are available in a public repository[2].

Code Evaluation: Static analysis of the four applications revealed a mean of 21.75 files and 269.75 functions, with an average cyclomatic complexity of 2.32 per function. SonarQube reported a mean of 2.25 bugs, 3.10 code smells per file, and 4.13% duplication density. No vulnerabilities or security hotspots were detected by the configured SonarQube rules. This result should not be interpreted as evidence that the applications are secure, since runtime vulnerabilities, application-level logic flaws, and potential data leakage were outside the scope of the analysis.

Feature Evaluation: The four applications contained 69 evaluated features, of which 46.38% were *Not Implemented* when pooled across all features. This differs from the macro-averaged MFR of 0.52 in Table 1, which gives equal weight to each application. Failures were particularly associated with multi-step workflows and functionality requiring persistent state or continuously available backend services. Thus, although MALFRA generated compilable mobile clients, its current architecture was insufficient for substantial backend-dependent behavior, showing that compilation alone is a poor proxy for functional completeness.

[1] https://flutter.dev/

[2] https://doi.org/10.5281/zenodo.22088681

Table 1. Macro-averaged descriptive statistics of feature evaluation across the four generated applications.

Metric	Mean	Std. Dev.	Median	Min	Max
FCR	0.31	0.31	0.25	0.00	0.76
FC	0.34	0.34	0.25	0.00	0.88
PIR	0.17	0.17	0.16	0.00	0.35
MFR	0.52	0.36	0.47	0.12	1.00

5. Conclusion

This study evaluated an automated pipeline for generating mHealth applications from high-level specifications using LLMs. Although all generated applications compiled, 46.38% of the features were not implemented, with failures concentrated in multi-step and backend-dependent functionality. These results reinforce that compilation and static code quality alone are insufficient indicators of successful end-to-end mHealth generation. Future work will focus on comparative baselines, backend-service generation, and evaluation with healthcare professionals.

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. This work is also partially supported by Propesq-UFRGS.

References

Alharbi, M. and Alshayeb, M. (2026). Automatic code generation techniques: A systematic literature review. *Automated Software Engineering*, 33(1):4.

Brandon, C. and Margaria, T. (2023). Low-code/no-code artificial intelligence platforms for the health informatics domain. *Electronic Communications of the EASST*, 82.

Joel, S., Wu, J., and Fard, F. (2025). A survey on llm-based code generation for low-resource and domain-specific programming languages. *ACM Trans. Softw. Eng. Methodol.*

Liu, S., La, H., Willms, A., and Rhodes, R. E. (2022). A "no-code" app design platform for mobile health research: Development and usability study. *JMIR formative research*, 6(8):e38737.

Niemiec, W., Tavares, A. R., and Cota, É. F. (2026). A methodology for generating mobile applications through large language models. In *2026 IEEE/ACM 13th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE. In press.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., et al. (2024). Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First conference on language modeling*.