ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/25nov074

LLM-Powered Legacy Code Modernization

Sahil Sanas[1]; Arya Naik[2]; Aditya Veerkar[3]; Pooja T. Kohok[4]

[1,2,3] Student, Pune Institute of Computer Technology
[4] Assistant Professor, Pune Institute of Computer Technology

Publication Date: 2025/11/10

Abstract: Legacy COBOL and C systems are still widely used in industries, yet they are increasingly costly, insecure, and incompatible with modern platforms. Traditional modernization methods, although effective, often require significant time and resources and carry high risks of disruption.

This paper proposes a scalable alternative that leverages Large Language Models (LLMs) within a structured multi-agent framework, guided by the "7Rs of Modernization." The frame- work comprises three agents: an Analysis Agent that interprets and maps legacy code, a Coder Agent that generates modern equivalents, and a Review Agent that validates correctness, security, and compliance through iterative feedback.

By automating much of the migration process, the proposed approach enables faster, more transparent, and less risky mod- ernization. It helps enterprises transition from outdated systems to modular, secure, and cloud-ready solutions, offering a cost- effective and future-proof pathway to digital transformation.

Keywords:- *LLM, Legacy Code, Modernization, 7Rs of Mod- Ernization, Multi-Agent Framework, Software Migration, Reliabil-Ity, Performance.*

How to Cite: Sahil Sanas; Arya Naik; Aditya Veerkar; Pooja T. Kohok (2025). LLM-Powered Legacy Code Modernization. *International Journal of Innovative Science and Research Technology*, 10(11), 118-121. https://doi.org/10.38124/ijisrt/25nov074

I. INTRODUCTION

Legacy systems, though still powering critical industries such as banking, insurance, and government, have become increasingly difficult to sustain in today's technology land- scape. Most of these systems were written decades ago in aging languages like COBOL or C, designed for hardware and business needs that no longer exist. Over time, this has created several pressing challenges.

A. *High Costs*

Maintaining these massive and tightly coupled codebases requires enormous financial investment. In fact, studies show that maintenance alone can consume up to 60% of an en- terprise's IT budget. The sheer size and complexity of these systems make even minor updates both time-consuming and expensive [3].

B. *Security and Performance Risks*

Built on outdated paradigms, legacy software lacks the modern safeguards needed to withstand cyber threats today. Their rigid procedural structure also hampers performance, making them more vulnerable to inefficiencies and system failures [5].

C. *Incompatibility with Modern Technology*

Legacy applications are rarely compatible with cloud com- puting, APIs, or modular, object-oriented designs. This incom- patibility prevents organizations from integrating their systems with modern platforms, limiting innovation and scalability [3].

- *Workforce Decline*

A critical concern is the steadily declining number of developers proficient in legacy languages such as COBOL. Recent estimates indicate that only around 5% of current programmers have experience in these systems, leaving many organizations without the necessary skills to maintain and modernize them effectively [3].

Together, these challenges create a vicious cycle of in-creased costs, increased risks, and architectural decay. Without modernization, enterprises risk being locked into fragile sys- tems that cannot keep pace with today's evolving business and technology demands [3], [5].

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/25nov074

II. LITERATURE SURVEY

Despite their long-standing presence, COBOL systems con- tinue to serve as the foundational infrastructure for the ma- jority of financial institutions worldwide [3]. However, these legacy systems are inherently complex, costly to maintain, and prone to operational inefficiencies and errors. To address these challenges, recent research has focused on artificial intelligence (AI)-driven approaches for the automated mod- ernization of COBOL applications into contemporary object- oriented languages such as Java [5]. Among these, a prominent method utilizes long short-term memory (LSTM) networks to parse COBOL code into Abstract Syntax Trees (ASTs) and subsequently translate them into optimized and maintainable Java code [1], [2]. This approach has demonstrated notable effectiveness, achieving up to 93% translation accuracy and a 35% improvement in maintainability, while ensuring the preservation of core business logic [3].

A. Statistical Analysis

Studies highlight the scale and urgency of the modernization challenge:

➢ *Prevalence & Impact:*

Approximately 70% of financial institutions still operate in COBOL systems, handling an estimated $3 trillion in daily transactions. Globally, more than 200 billion lines of COBOL code remain in active use. Maintaining these systems consumes up to 60% of IT budgets, while the COBOL workforce has decreased to just 5% of developers [3].

➢ *Technical Debt & Complexity:*

On average, COBOL modules have 18 decision paths (compared to the modern ideal of 10) and 8 dependencies per module, leading to a 30% higher defect rate compared to modern languages such as Java [5].

➢ *Challenges of Existing Solutions:*

Manual modernization achieves only 75% accuracy, re- quires six months for every 10,000 lines of code, and risks losing 40% of business logic. Rule-based tools (e.g., Micro Focus) improve accuracy to about 82%, but still miss contextual nuances in 20% of cases [3], [5].

➢ *AI-Based Results:*

In comparison, AI-driven approaches achieve 93% accu- racy, improve maintainability by 35%, reduce complexity from 18 to 9, lower coupling from 8 to 4, and retain 93% of the original logic [3].

➢ *Market & Context:*

Legacy modernization is expected to reach a $500 billion market, with 62% of developers seeking AI tools for support. Since 80% of enterprise applications run on Java, it is the primary focus for migration [3].

➢ *Scale of AI-Driven Migrations:*

Solovyeva et al. demonstrate migration of a 2.5M-line PL/SQL system into Java using domain-model guided prompting. Their results show high syntactic correctness and promising functional accuracy [6].

➢ *Benchmark Performance:*

In the CODEMENV dataset, comprising 922 migration examples across 19 Python/Java packages, the average pass@1 rate across LLMs was just 26.50%, with the strongest model (GPT-4O) reaching 43.84%. This gap shows that while LLMs are useful, their outputs remain unreliable without human validation [7].

B. Modernization Tools and Approaches

➢ *IBM Watsonx Code Assistant for Z (WCA4Z):*

IBM's WCA4Z is designed for COBOL-to-Java migra- tion. Instead of simple line-by-line conversion, which often produces unmaintainable "JOBOL" (Java code that mimics COBOL logic), it follows a two-phase process: first, a Class Designer generates the Java class structure; then method-level transformations are applied with hu- man oversight to ensure accuracy and maintainability [5].

➢ *Evaluation of Translation:*

The quality of translated code is assessed through:

- *Syntactic Checking* –

Ensures generated Java code compiles without common LLM errors such as re- peated loops [5].

- *Semantic Checking* –

Verifies correct handling of variables, procedures, and middleware (e.g., CICS or SQL) [5].

- *LLMs as Judges (LaaJs)* –

Other LLMs evaluate the translation, focusing on logic preservation and idiomatic correctness [5].

➢ *LSTM Networks:*

Long Short-Term Memory (LSTM) networks are effective for code translation as they capture long-term depen- dencies in sequential data, addressing the limitations of traditional RNNs. Their three gates—Forget, Input, and Output—allow selective retention, updating, and passing of information, making them well-suited for handling legacy code [1], [2].

➢ *Multi-agent upgrades*

Ala-Salmi et al. proposed an autonomous multi-agent pipeline for upgrading legacy web applications. By dis- tributing roles such as file updater and view upgrader across agents, their method reduced translation errors and improved precision compared to single-agent promptings [10].

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/25nov074

➢ *Domain-specific improvements:*

In the COBOL/mainframe domain, Dau et al. introduced XMainframe, a model designed for modernization tasks. On MainframeBench, it achieved 30% higher accuracy on multiple-choice QA than DeepSeek-Coder, doubled the BLEU score of Mixtral-Instruct 8x7B on QA tasks, and scored six times higher than GPT-3.5 for COBOL summarization [8].

➢ *Industrial-scale adoption:*

At Google, Ziftci et al. reported on 39 migrations conducted over 12 months by three developers. These migrations involved 595 code changes and 93,574 total edits. Of these, 74.45% of the code changes and 69.46% of edits were generated by an LLM, resulting in an estimated 50% reduction in migration time compared to prior manual efforts [9].

III. PROPOSED SOLUTION

To address the challenges of legacy system modernization, we propose a novel approach that leverages Large Language Models (LLMs) within a structured multi-agent framework. The solution is guided by the well-established concept of the "7Rs of Modernization" and employs advanced custom fine-tuned models.

The framework is built around three specialized agents, each with a distinct role:

A. Analysis Agent:

Acts as the architect of the project. Using static analysis tools, it maps out the dependency tree, interfile relationships, and logical flow of the legacy codebase. Its main responsibility is to generate a clear modernization roadmap, breaking down the migration process into smaller, well-structured tasks arranged in the correct order.

B. Coder Agent:

Once tasks are defined, the Coder Agent translates them into modern code. It uses fine-tuned LLMs to ensure high-fidelity code generation, producing secure, optimized, and functionally accurate implementations. If required, it can also handle additional subtasks defined during the process.

C. Review Agent:

After code is generated, the Review Agent evaluates its quality, accuracy, and security. It checks for vulnerabilities, ensures that the new implementation is in accordance with modern standards, and verifies that the intended functionality is preserved.

A key strength of this framework is the **feedback loop**. If the Review Agent identifies issues, it sends structured feedback to the Analysis Agent, which then creates a refined task for the Coder Agent. This iterative process ensures continuous improvement, reduces the need for heavy human supervision, and allows external inputs when needed.

By distributing responsibilities among specialized agents and embedding feedback into the workflow, the modernization process becomes more transparent, manageable, and reliable. This approach minimizes risks while ensuring that the final migrated system is both modern and robust.

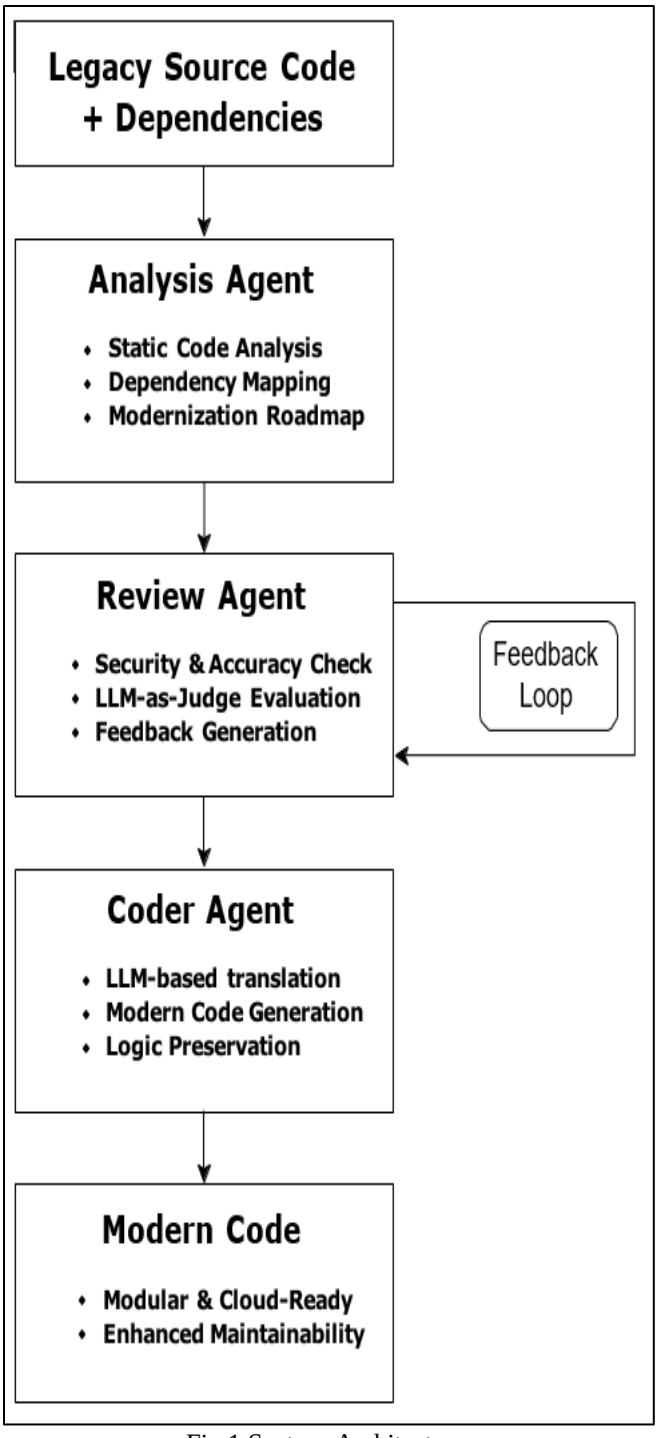

Fig 1 System Architecture

ISSN No: -2456-2165 https://doi.org/10.38124/ijisrt/25nov074

IV. CONCLUSION

Legacy systems have long supported critical industries, but their growing maintenance costs, security vulnerabilities, and incompatibility with modern platforms make their continued reliance unsustainable. Traditional modernization approaches are often too slow, risky, and resource-intensive to meet present-day demands. The emergence of Large Language Models offers new opportunities, but the evidence highlights both potential and limitations. Large-scale deployments at Google show that over two-thirds of edits can be automated, with an estimated 50% reduction in migration time [9]. At the same time, benchmarks like CODEMENV reveal that LLMs still solve fewer than half of migration tasks correctly [7]. Domain-specific models such as XMainframe demonstrate that targeted training can deliver dramatic improvements in special- ized contexts like COBOL [8].This paper presented a novel multi-agent framework powered by Large Language Mod- els to address these challenges. By dividing responsibilities among Analysis, Coder, and Review agents and incorporating a feedback loop, the approach ensures structured, accurate, and efficient code migration. The result is a practical pathway for enterprises to transform outdated software into secure, modular, and cloud-ready systems while preserving essential functionality.

REFERENCES

[1]. K. Greff, R. K. Srivastava, J. Koutn´ık, B. R. Steunebrink, and J. Schmidhuber, “LSTM: A search space odyssey,” IEEE Transactions on Neural Networks and Learning Systems, vol. 28, no. 10, pp. 2222–2232, Oct. 2016.

[2]. S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural Computation, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[3]. G. Bandarupalli, “Code Reborn: AI-Driven Legacy Systems Modern- ization from COBOL to Java,” arXiv preprint arXiv:2504.11335, Apr. 2025. Available: https://arxiv.org/abs/2504.11335

[4]. J. Fitzpatrick, “Case Study: Converting C Programs to C++,” C++ Report, vol. 8, no. 2, p. 40, 1996.

[5]. S. Froimovich, R. Gal, W. Ibraheem, and A. Ziv, “Quality Eval- uation of COBOL to Java Code Transformation,” arXiv preprint arXiv:2507.23356, Jul. 2025. Available: https://arxiv.org/abs/2507.23356

[6]. L. Solovyeva, *et al.*, “Leveraging LLMs for Automated Translation of Legacy Code: A Case Study on PL/SQL to Java Transformation” arXiv:2508.19663v1 [cs.SE] 27 Aug 2025. Available: https://arxiv.org/ html/2508.19663

[7]. K. Cheng, X. Shen, Y. Yang, T. Wang, Y. Cao, M. A. Ali, H. Wang, L. Hu, D. Wang, “CODEMENV: Benchmarking Large Language Models on Code Migration,” arXiv preprint arXiv:2506.00894, 2025. Available: https://arxiv.org/abs/2506.00894

[8]. A. T. V. Dau, H. T. Dao, A. T. Nguyen, *et al.*, “XMainframe: A Large Language Model for Mainframe Modernization,” arXiv preprint arXiv:2408.04660, 2024. Available: https://arxiv.org/abs/2408.04660

[9]. C. Ziftci, S. Nikolov, A. Sjo¨vall, B. Kim, D. Codecasa, M. Kim, *et al.*, “Migrating Code At Scale With LLMs At Google,” arXiv preprint arXiv:2504.09691, 2025. Available: https://arxiv.org/pdf/2504.09691

[10]. J. Ala-Salmi, *et al.*, “Autonomous Multi-Agent Modernization of Legacy Web Applications,” arXiv preprint arXiv:2501.19204, 2025. Available: https://arxiv.org/abs/2501.19204

[11]. C. Diggs, M. Doyle, A. Madan, S. Scott, E. Escamilla *et al.*, “Leveraging LLMs for Legacy Code Modernization: Challenges and Opportunities for LLM-Generated Documentation” arXiv preprint arXiv:2411.14971, 2024. Available: https://arxiv.org/abs/2411.14971