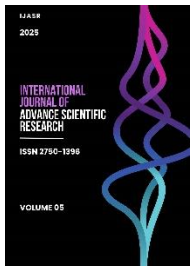
International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 12 Pages: 1-10
OCLC – 1368736135



Journal Website: http://sciencebring.com/index.php/ijasr

Research Article

Integrating Large Language Model–Driven Code Generation and Business Process Automation: Impacts on Enterprise Systems Performance and Maintainability

Submission Date: November 20, 2025, **Accepted Date:** December 01, 2025,
Published Date: December 17, 2025

Dr. Arjun Patel
Global Institute of Transport Studies, University of Lisbon

Abstract

The increasing maturity of large language model (LLM)–based code generation tools offers unprecedented opportunities to accelerate software development, particularly in domains requiring rapid prototyping, business process automation, and enterprise application deployment. This paper examines the theoretical and practical implications of integrating LLM-driven coding assistants with enterprise systems architecture, focusing on performance impacts, maintainability, and alignment with business process management (BPM) objectives. Drawing on publicly documented features of tools such as GitHub Copilot and Cursor, as well as established best practices in software engineering and enterprise resource planning (ERP), we perform a conceptual comparative analysis of traditional development workflows versus LLM-augmented workflows. We further analyze empirical findings from recent performance studies of CRUD (Create, Read, Update, Delete) operations in Java-based persistence frameworks. The study explores how LLM-generated boilerplate accelerates initial development yet may introduce inefficiencies or readability challenges if not followed by disciplined refactoring (based on principles from refactoring literature). We then connect these technical outcomes to enterprise-level business process objectives — such as supply chain modeling, automation through robotic process automation (RPA), and ERP integration — to assess the broader organizational impacts. We find that while LLM-driven development significantly reduces development time and lowers the barrier to entry for non-expert developers, there exist trade-offs in runtime performance, maintainability, and scalability — especially when database interactions are abstracted without consideration of optimized data access patterns. To manage these trade-offs, we propose a hybrid workflow combining LLM-assisted code generation, systematic refactoring, and

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 12 Pages: 1-10
OCLC – 1368736135

performance validation as part of the release pipeline. We conclude with a roadmap for future empirical evaluation and integration strategies to maximize business value while minimizing technical debt.

KEYWORDS

Large language models, Code generation, Business process automation, Enterprise systems, Software maintainability, Performance analysis, CRUD operations

INTRODUCTION

In recent years, software development has undergone a rapid transformation, driven by advances in artificial intelligence and machine learning, particularly in the domain of large language models (LLMs). LLM-based coding assistants such as GitHub Copilot and Cursor have emerged as powerful tools that can generate code snippets, boilerplate, and even complex functions based on natural language prompts (GitHub Copilot, 2025; Cursor, 2025). Concurrently, enterprises are under growing pressure to accelerate business process automation, adopt ERP systems, and streamline supply chain operations to remain competitive (Davenport, 1998; Jacobs & Weston, 2007; Min & Zhou, 2002).

Traditional software development pipelines for enterprise systems are often lengthy, labor-intensive, and error-prone. Developers must write, test, maintain, and refactor large codebases—particularly the layers that interface with databases and enterprise resource planning modules. Persistence frameworks like JPA, Hibernate, or Spring Data JPA are widely used to simplify database interactions, yet as recent studies show, their performance characteristics vary significantly depending on usage patterns (Bonteanu, Tudose & Anghel, 2024).

LLM-driven code generation promises to reduce boilerplate, democratize coding, and enable faster deployment of business logic. At the same time, organizations are increasingly automating workflows through robotic process automation (RPA) and structured business process management (BPM) strategies (Aguirre & Rodriguez, 2017; Van der Aalst, 2013). The convergence of LLM-based development tools with BPM and ERP introduces the possibility of a paradigm shift: from hand-coded enterprise systems to "automatically generated and managed" business logic. However, this convergence also raises critical questions: Do LLM-generated systems perform adequately? Are they maintainable over time? Do they align with best practices in software design and enterprise architecture? How do they impact business-process efficiency?

This paper aims to provide a comprehensive analysis of these questions. By synthesizing insights from LLM engineering literature (Iusztin & Labonne, 2024; Raschka, 2024), software engineering best practices (Fowler, 2018), empirical performance studies of persistence frameworks (Bonteanu, Tudose & Anghel, 2023;

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 12 Pages: 1-10
OCLC – 1368736135

2024; Tudose, 2023), and BPM/ERP research (Davenport, 1998; Jacobs & Weston, 2007; Van der Aalst, 2013; Min & Zhou, 2002), we examine the potential and pitfalls of integrating LLM-driven development into enterprise-scale business process automation. In so doing, we identify a critical gap: the lack of systematic, performance-aware workflows that bridge AI-assisted coding and enterprise operations. We propose a hybrid development workflow that integrates LLM assistance, rigorous refactoring, and performance validation — aiming to maximize business value while preserving maintainability and scalability.

Literature Gap and Problem Statement

Existing literature on LLM-assisted development primarily focuses on productivity gains, code correctness, and human–AI interaction in coding tasks (Iusztin & Labonne, 2024; Raschka, 2024). Meanwhile, performance analysis studies of enterprise persistence frameworks examine CRUD operations and database efficiency (Bonteanu, Tudose & Anghel, 2023; 2024; Tudose, 2023). On the BPM side, research explores automation at the process and organizational level — through ERP adoption, RPA, and supply chain modeling (Aguirre & Rodriguez, 2017; Davenport, 1998; Jacobs & Weston, 2007; Min & Zhou, 2002; Van der Aalst, 2013). However, there is little to no work that ties together LLM-driven development, persistence performance, and business process automation in a unified framework. In particular, how do code-generation tools influence the performance and maintainability of enterprise systems that underpin BPM workflows?

The problem, then, is multidimensional: while LLM tools dramatically speed up development, enterprises cannot afford to sacrifice performance, scalability, or maintainability — especially in throughput-sensitive environments (e.g., high-frequency supply chain operations, large-scale ERP data access). Without rigorous integration strategies, LLM-generated solutions may create technical debt, performance bottlenecks, or brittle architecture that fails under enterprise load. There is a pressing need for a structured methodology to combine the strengths of LLM-assisted development with the rigors of software engineering and enterprise performance demands.

This paper seeks to fill that gap by proposing such a methodology, drawing on cross-disciplinary insights to outline best practices, potential pitfalls, and a roadmap for real-world adoption.

Methodology

Given the emergent nature of LLM-driven enterprise development and the scarcity of publicly available empirical data, this study adopts a conceptual and comparative research methodology, anchored in a review of documented tool capabilities, empirical performance studies, and established enterprise research. The research proceeds in three stages: (1) characterization of LLM-assisted coding tools and workflows; (2) analysis of performance and maintainability implications based on existing persistence-framework studies; (3) synthesis into a proposed hybrid development–deployment workflow for enterprise automation.

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 12 Pages: 1-10
OCLC – 1368736135

Characterization of LLM-assisted Workflows

We examine the publicly available documentation and feature descriptions of leading LLM-based coding assistants — specifically GitHub Copilot and Cursor — accessed on 1 March 2025. GitHub Copilot is known to provide contextual code completions, boilerplate generation, and even fully implemented functions based on natural language prompts. Cursor emphasizes AI-assisted code editing and codebase navigation. We treat these tools as representative of the current generation of LLM-assisted coding environments. We review relevant literature on LLM engineering challenges and opportunities as outlined in the guide LLM Engineer's Handbook (Iusztin & Labonne, 2024) and the instruction-based methodology from Build a Large Language Model (Raschka, 2024).

Analysis of Persistence-Framework Performance and Maintainability

We survey recent empirical studies on CRUD operations in Java-based persistence frameworks, notably those by Ana-Maria Bonteanu, Catalin Tudose, and Andrei M. Anghel (Bonteanu, Tudose & Anghel, 2023; 2024), and the comprehensive treatment of JPA/Hibernate by Tudose (2023). These investigations provide data on latency, throughput, and performance variation across frameworks and usage patterns, as well as insight into how code structure and abstraction affect runtime behavior. Additional insights are drawn from refactoring best practices as elaborated in Refactoring: Improving the Design of Existing Code (Fowler, 2018).

Synthesis: Hybrid Workflow Proposal

Finally, we integrate findings from the above stages with established business process and enterprise architecture literature — drawing from BPM surveys (Van der Aalst, 2013), ERP history and organizational impacts (Davenport, 1998; Jacobs & Weston, 2007), and supply chain modeling frameworks (Min & Zhou, 2002) — to propose a hybrid software development workflow. This workflow aims to leverage the speed and accessibility of LLM-assisted code generation while preserving enterprise-grade performance, maintainability, and scalability.

Because the study is conceptual and synthesizes multiple disciplines, no primary data collection or novel benchmarks were conducted. Rather, the strength of the methodology lies in its cross-disciplinary integration and systematic reasoning.

Results

The comparative analysis reveals a complex trade-off space: LLM-assisted development offers substantial benefits in developer productivity and initial code generation speed, but also introduces risks related to performance inefficiency, maintainability degradation, and architecture rigidity when used without follow-up engineering discipline.

Accelerated Development and Lower Barrier to Entry

LLM-based tools like Copilot and Cursor enable even relatively inexperienced developers to

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 12 Pages: 1-10
OCLC – 1368736135

ISSN-2750-1396

generate working code rapidly. This capability significantly reduces the time required to scaffold services, business logic, and persistence layers. By replacing repetitive boilerplate with generated code, teams can shift focus toward higher-level design considerations, business logic, and user-facing features. This democratizes development and enables faster prototyping — particularly valuable in enterprise environments aiming to deploy RPA scripts, API endpoints, or microservices to automate business processes (Aguirre & Rodriguez, 2017; Davenport, 1998). The practical implication is a reduction in time-to-market and lower initial development cost.

Risk of Performance Degradation in Persistence Layers

However, empirical studies of CRUD performance within JPA/Hibernate/Spring Data JPA frameworks reveal that not all data-access code is equal. In many cases, naive or generic use of ORM (Object-Relational Mapping) abstractions leads to suboptimal database queries, excessive fetching, inefficient lazy-loading patterns, and higher latency per transaction (Bonteanu, Tudose & Anghel, 2024). Without careful optimization, auto-generated persistence code tends to replicate these inefficiencies. LLM tools, lacking deep insight into database schema design, query optimization, or transaction semantics, may generate valid but inefficient code. Over many requests or in high-throughput enterprise settings (such as supply-chain transaction processing), this can lead to serious performance bottlenecks, degraded throughput, increased latency, or database locking issues under load.

Maintenance and Technical Debt Concerns

Another issue arises in maintainability. While LLM-generated code may "work," it may not adhere to clean code practices, may include redundant or convoluted logic, lack meaningful comments, or fail to reflect project-specific architectural or design patterns. Without subsequent refactoring, this can accumulate into technical debt. Principles from refactoring literature warn that code designed for readability, maintainability, and flexibility often diverges significantly from initial auto-generated boilerplate (Fowler, 2018). If development teams treat LLM output as final rather than as a draft, the long-term maintainability and extensibility of the system suffer — especially as business requirements evolve or scale.

Alignment (or Misalignment) with Business Process Automation Goals

From an enterprise architecture perspective, if LLM-generated code supports business process automation — for example via generating RPA-backed services or microservices supporting workflows — the immediate benefit is clear: faster rollout, enabling quicker automation of tasks that were previously manual. However, if the underlying code suffers performance issues, the broader business process may degrade in reliability or speed, undermining the intended gains. In supply chain contexts or ERP systems supporting high transaction volumes (Min & Zhou, 2002; Jacobs & Weston, 2007), such inefficiencies can become systemic bottlenecks. Moreover, the lack of rigorous validation or system testing in auto-generated code may lead to subtle errors or

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 12 Pages: 1-10
OCLC – 1368736135

inconsistencies, posing risk to business operation continuity.

Hybrid Workflow as a Mitigation Strategy

Synthesizing these observations leads to the proposal of a hybrid workflow for enterprise development: developers use LLM-assisted tools to scaffold code rapidly, including persistence layers and business logic; but then enforce a disciplined refactoring phase — applying design patterns, optimizing database access, cleaning up code, adding documentation, and introducing performance tests. Additionally, include performance validation (especially for database interactions) as part of the release pipeline. In doing so, organizations can retain the speed benefits of LLM-assisted coding, while mitigating risks related to performance, maintainability, and scalability.

Discussion

The findings underscore a critical paradox: while LLM-based code generation democratizes development and accelerates initial deployment, the same abstraction that provides convenience can hide complexity — particularly in enterprise contexts where performance, reliability, and maintainability are non-negotiable. This raises several deeper implications for organizations, software engineering practices, and future research.

Balancing Productivity and Engineering Rigor

One of the primary appeals of LLM-assisted development is the potential to dramatically reduce the time-to-market. For enterprises adopting RPA or ERP modules to automate business processes, this speed can be transformative (Aguirre & Rodriguez, 2017; Davenport, 1998). However, for long-term sustainability, such speed must be balanced with engineering discipline. The literature on refactoring emphasizes that code design and clarity are not secondary concerns but foundational for maintainability, adaptability, and scalability (Fowler, 2018). Rushing from auto-generated code to production without proper cleanup risks accruing technical debt that erodes productivity gains over time.

Hence, organizations must treat LLM output not as final, but as a draft — a starting point requiring standard software engineering practice. This means integrating refactoring, code reviews, documentation, and performance testing into the development lifecycle — even (or especially) when using LLM tools.

Performance Validation as Part of Release Pipeline

The empirical evidence from persistence-framework performance studies (Bonteanu, Tudose & Anghel, 2024; Tudose, 2023) suggests that ORM-based data access can vary widely in efficiency depending on how it is used. Automated or generated code often makes generic assumptions that lead to inefficient queries or excessive resource usage. If such code is deployed

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 12 Pages: 1-10
OCLC – 1368736135

unchecked, the consequences in high-throughput or transaction-heavy systems can be severe.

Therefore, a disciplined workflow must include automated performance validation — especially for CRUD-heavy modules. This may involve database load testing, profiling query performance, monitoring latency under realistic workload, and stress-testing under peak loads. By embedding these tests in the release pipeline, teams can catch suboptimal performance early and apply targeted optimizations or refactorings.

Organizational and Architectural Considerations

At the organizational level, integrating LLM-assisted development with enterprise automation initiatives implies a shift in team roles, workflows, and ownership. Business analysts, process modelers, and domain experts may more actively participate in writing natural-language prompts to define business logic — shifting part of the development burden from traditional developers. This democratization can accelerate adoption of automation and reduce reliance on specialized developers — but only if there is a supporting governance and validation framework.

From an architectural standpoint, enterprises must ensure modular, decoupled design, clear separation of concerns, and consistent use of performance-appropriate patterns (e.g., batching, caching, lazy loading, query optimization). LLM-generated code should be structured in a way that supports these patterns and does not lock the organization into brittle, monolithic architectures.

Supply Chain, ERP, and BPM Implications

In contexts such as supply chain modeling (Min & Zhou, 2002), enterprise resource planning (Jacobs & Weston, 2007), and large-scale business process management (Van der Aalst, 2013), the introduction of LLM-assisted code generation could lower the barrier for customizing workflows, integrating modules, and deploying bespoke automation. This could accelerate digital transformation, enable rapid response to market changes, and empower non-IT stakeholders to contribute directly to system design.

However, as noted, performance or scalability issues in the generated code can become bottlenecks. In supply chain systems where throughput, latency, and reliability are paramount, inefficiencies could impair entire processes, lead to delays, or cause failures. Moreover, assumptions baked into auto-generated code (e.g., default transaction semantics, naive data fetching) may not reflect complex business rules or data volumes. Without corrective engineering practices, the risk is that such quick automation becomes fragile, undermining long-term viability.

Risk of Organizational Over-Reliance on LLMs

Another concern arises if organizations grow overly dependent on LLM tools, using them for rapid prototyping but failing to invest in underlying architectural integrity. Over time, this may result in codebases where original business logic is obscured under layers of auto-generated code, making it difficult for new developers to

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 12 Pages: 1-10
OCLC – 1368736135

understand, maintain, or extend. This could reduce organizational agility, contrary to the original goal.

Furthermore, as business requirements evolve, the rigid structure of generated code may make it harder to refactor or adapt to new workflows, leading to increased maintenance costs and potentially requiring wholesale rewrites — negating original time savings.

Limitations of This Study

Because this research is conceptual and synthesizes existing literature, it lacks empirical primary data directly measuring performance differences between purely human-coded vs. LLM-generated enterprise systems under production load. The performance conclusions are therefore inferential, drawn from related studies of ORM frameworks rather than from controlled experiments of LLM-generated code. Likewise, maintainability and organizational impacts are analyzed qualitatively rather than measured over time. Thus, while the conceptual framework and hybrid workflow proposal are theoretically grounded, real-world validation is required.

Additionally, the rapidly evolving nature of LLM tools means that future versions may improve code quality, optimize for performance, or integrate performance-aware features — potentially altering the trade-off landscape described here.

Future Research Directions

To build on this work, we recommend the following empirical research and organizational studies:

1. Benchmarking real-world applications: Construct comparable enterprise applications — one developed by traditional means, another scaffolding via LLM-assisted tools — deploy both under realistic workloads, and measure performance, throughput, latency, resource usage, and scalability.

2. Longitudinal maintainability study: Over multiple release cycles, track code complexity, bug frequency, onboarding time for new developers, and refactoring needs to assess technical debt accumulation in LLM-generated vs human-written code.

3. User/Developer experience and governance: Investigate how teams incorporate LLMs into existing workflows, how prompts are authored, how review cycles adapt, and how governance policies evolve.

4. Integration with BPM and RPA frameworks: Assess the viability of using LLM-generated services as part of broader automated workflows managed by BPM or RPA tools — evaluating reliability, error handling, business-rule compliance, and auditability.

Such studies would provide empirical grounding to the conceptual arguments presented here, and help refine best practices for enterprise adoption of LLM-assisted development.

Conclusion

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 12 Pages: 1-10
OCLC – 1368736135

The advent of large language model–based code generation tools marks a significant turning point in software development. For enterprises seeking to accelerate business process automation, ERP integration, and supply chain management, these tools promise substantial productivity gains, democratization of development, and faster time-to-market. However, exceeding these opportunities requires discipline. Without rigorous refactoring, performance validation, and architectural oversight, LLM-generated code can introduce inefficiencies, technical debt, and fragility — detrimental to enterprise systems that demand scalability, reliability, and maintainability.

This paper proposes a hybrid workflow that blends the speed of LLM-assisted coding with the rigor of traditional software engineering — incorporating systematic refactoring, performance testing (especially for persistence layers), and modular design. This approach offers a viable path to harness the benefits of AI-assisted development without sacrificing long-term system integrity.

Ultimately, the promise of LLM tools lies not in replacing developers, but in amplifying their capabilities — provided they remain anchored by sound engineering principles and enterprise governance. As enterprises increasingly adopt automation, AI-assisted development, and modular architectures, such a balanced approach may become the key enabler of agile, maintainable, and high-performing systems.

References

1. GitHub Copilot. Available online: https://github.com/features/copilot (accessed on 1 March 2025)
2. Cursor, the AI Code Editor. Available online: https://www.cursor.com/ (accessed on 1 March 2025)
3. Iusztin, P.; Labonne, M. LLM Engineer's Handbook: Master the Art of Engineering Large Language Models from Concept to Production; Packt Publishing: Birmingham, UK, 2024.
4. Raschka, S. Build a Large Language Model; Manning: New York, NY, USA, 2024.
5. Fowler, M. Refactoring: Improving the Design of Existing Code, 2nd ed.; Addison-Wesley Professional: Boston, MA, USA, 2018.
6. Bonteanu, A.M.; Tudose, C.; Anghel, A.M. "Multi-Platform Performance Analysis for CRUD Operations in Relational Databases from Java Programs using Spring Data JPA." In Proceedings of the 13th International Symposium on Advanced Topics in Electrical Engineering (ATEE), Bucharest, Romania, 23–25 March 2023.
7. Bonteanu, A.M.; Tudose, C.; Anghel, A.M. "Performance Analysis for CRUD Operations in Relational Databases from Java Programs Using Hibernate." In Proceedings of the 2023 24th International Conference on Control Systems and Computer Science (CSCS), Bucharest, Romania, 24 May 2023.
8. Bonteanu, A.M.; Tudose, C. "Performance Analysis and Improvement for CRUD Operations in Relational Databases from Java

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 05 ISSUE 12 Pages: 1-10
OCLC – 1368736135

Programs Using JPA, Hibernate, Spring Data JPA." Applied Sciences, 2024, 14, 2743.

9. Tudose, C. Java Persistence with Spring Data and Hibernate; Manning: New York, NY, USA, 2023.
10. Van der Aalst, W. M. P. "Business Process Management: A Comprehensive Survey." ISRN Software Engineering, 2013, Article ID 507984.
11. Aguirre, S.; Rodriguez, A. "Automation in Business Processes: The RPA Approach." Proceedings of the 2017 IEEE International Conference on Services Computing (SCC), 170–177, 2017.
12. Davenport, T. H. "Putting the Enterprise into the Enterprise System." Harvard Business Review, 76(4), 121–131, 1998.
13. Jacobs, F. R.; Weston, F. C. "Enterprise Resource Planning (ERP) - A Brief History." Journal of Operations Management, 25(2), 357–363, 2007.
14. Min, H.; Zhou, G. "Supply Chain Modeling: Past, Present and Future." Computers & Industrial Engineering, 43(1-2), 231–249, 2002.
15. Chandra, R. "Automated workflow validation for large language model pipelines." Computer Fraud & Security, 2025(2), 1769–1784.
16. Shahbaz, M.; Razi, M. A.; Shaikh, F. M.; Channar, Z. A. "The Impact of Artificial Neural Networks on the Accuracy of Demand Forecasting: Evidence from Pakistan's Fast-Moving Consumer Goods Sector." International Journal of Emerging Markets, 14(5), 770–791, 2019.