**FMDB** *Transactions on Sustainable Computing Systems*

# Next-Generation Code Transformation for Legacy .NET Systems with Generative AI

**Hema Latha Boddupally**[1,*]

[1]Department of Technical Architect, Information Technology, R3 Technology Inc, Hillsborough, New Jersey, United States of America.
hemalatha607@gmail.com[1]

**Abstract:** Generative AI-powered code transformation techniques for legacy .NET systems address the long-standing challenges encountered by organisations that depend on ageing application stacks, tightly coupled components, and outdated development models that limit scalability, maintainability, and integration with modern architectures. The objective of this study is to examine whether generative AI can provide a structured and dependable pathway for modernising complex .NET applications while reducing the extensive manual effort traditionally required for refactoring and architectural redesign. The research problem centres on assessing the ability of generative AI to interpret the semantics of legacy code, propose optimised transformations, and preserve functional accuracy throughout the modernisation process. A mixed-methods approach was used, incorporating quantitative evaluation of code quality gains, transformation precision, and performance improvements, alongside qualitative analysis of developer experience, maintainability, and architectural alignment. Key findings show that generative AI accelerates modernisation workflows, enhances consistency across transformed modules, and supports the transition of legacy logic to modular, cloud-oriented, and testable designs. The study contributes strategically by presenting a structured framework for integrating AI-assisted refactoring into enterprise modernisation initiatives and academically by offering a comprehensive assessment of generative AI capabilities in legacy system evolution. The results indicate that generative AI can act as a powerful catalyst when combined with human oversight, validation, and governance.

**Keywords**: Generative AI; Code Transformation; Legacy .NET Systems; Automated Software Engineering; Structured Framework; Modernisation Workflows; Cloud-Oriented; Testable Designs.

**Received on:** 13/01/2025, **Revised on:** 09/04/2025, **Accepted on:** 08/06/2025, **Published on:** 09/12/2025

**Journal Homepage:** https://www.fmdbpub.com/user/journals/details/FTSCS

**DOI:** https://doi.org/10.69888/FTSCS.2025.000527

**Cite as:** H. L. Boddupally, "Next-Generation Code Transformation for Legacy .NET Systems with Generative AI," *FMDB Transactions on Sustainable Computing Systems*, vol. 3, no. 4, pp. 253-263, 2025.



## 1. Introduction

The legacy .NET systems remain central to enterprise operations, yet many were built using architectural patterns, frameworks, and development practices that no longer align with current software engineering demands [1]. Over time, such systems accumulate complexity and technical debt, making it increasingly difficult to adapt to new business requirements. Organisations have long recognised the need to modernise these systems [2]. Still, conventional approaches often involve intensive manual

[*]Corresponding author.

effort and substantial risk, especially when core functionality is poorly documented or deeply embedded in monolithic structures [3]. Generative artificial intelligence has emerged as a transformative force within software engineering, offering capabilities that extend well beyond traditional automation. With its ability to interpret code semantics, infer architectural context, and generate new code constructs, generative AI introduces a compelling opportunity to reimagine modernisation practices for legacy .NET environments. These capabilities pave the way for more efficient transformation of large codebases that would otherwise require significant specialised expertise and prolonged modernisation cycles [4].

Despite these advancements, the systematic application of generative AI to legacy .NET modernisation remains insufficiently explored in both industry and academic settings. Many organisations still rely heavily on manual refactoring techniques because the reliability, accuracy, and scalability of AI-assisted transformation approaches have not been thoroughly validated [5]. This lack of consolidated research creates a clear gap that hinders broader adoption and raises uncertainty about the practical readiness of generative AI in this domain. The core problem addressed in this study centres on determining whether generative AI can effectively support the transformation of legacy .NET systems while maintaining functional correctness and architectural coherence. The motivation arises from the need to overcome limitations associated with manual modernisation, which often struggles to balance precision, speed, and cost-effectiveness [6]. As generative AI becomes increasingly capable of reasoning about software structures, it becomes essential to examine how these advancements can be harnessed responsibly and reliably for real-world modernisation scenarios. The primary objectives of this research include assessing the capabilities of generative AI in analysing legacy .NET codebases, identifying suitable transformation techniques, and generating optimised code alternatives that align with modern software engineering principles [7]. The research further seeks to evaluate the feasibility of using AI-generated transformations in production-scale environments and to identify best practices for integrating these capabilities into existing development workflows. To guide the study, several research questions are formulated: how effectively generative AI models can understand the semantics of legacy code, which categories of transformations they can automate, and how these transformations impact system quality, maintainability, and long-term sustainability [8].

Additional questions address the degree of human oversight required and the mechanisms necessary to ensure that AI recommendations remain trustworthy and consistent across diverse legacy environments [9]. The significance of this work lies in its potential to expand modernisation strategies for enterprises that rely on long-standing .NET systems. By offering insights into practical applications of generative AI, the study equips industry professionals with knowledge that can reduce modernisation costs, shorten transformation timelines, and enhance the reliability of code restructuring processes [10]. These outcomes have direct implications for digital transformation initiatives, cloud migration programs, and long-term technology evolution planning [11]. From an academic perspective, this study contributes to the growing field of AI-driven software engineering by providing a structured and empirical examination of how generative AI can facilitate software evolution. It advances theoretical understanding of AI-enabled code transformation and offers a foundation for future research focused on automated refactoring, architecture recovery, and intelligent software maintenance [12]. Overall, this introduction establishes the need for a comprehensive, methodical investigation of generative AI-powered code-transformation techniques for legacy .NET systems. By addressing both the practical challenges and academic gaps, the study positions generative AI as a promising catalyst for next-generation modernisation and sets the stage for deeper exploration throughout the remainder of the work [13].

## 2. Foundational Theories and Prior Research

Research on generative artificial intelligence for software engineering has expanded significantly, with early studies demonstrating how language models trained on code can analyse structures, infer semantics, and generate syntactically consistent program fragments [14]. These foundational contributions established the theoretical basis for the use of generative models to support software maintenance and evolution. Within this body of work, a recurring theme is the ability of generative models to learn patterns from large code corpora, enabling them to perform tasks such as code completion, translation, and summarisation with increasing accuracy [15]. These advancements provide a conceptual anchor for exploring their applicability to modernisation tasks in legacy .NET systems, where semantic understanding and transformation of large codebases are essential. Relevant frameworks in the literature include neural encoder-decoder architectures, graph-based code representation models, and attention-based transformers. These frameworks emphasise structural learning, context awareness, and multi-level reasoning about code artefacts, all of which are essential for effective transformation. Studies adopting these frameworks highlight the importance of representing code not only as text but also as an interconnected graph of operations and dependencies [16]. These insights inform the design of generative AI techniques capable of refactoring or restructuring legacy systems by capturing both syntactic and semantic relationships within .NET applications.

Prior academic work has also contributed models for automated bug fixing, program synthesis, and cross-language translation, demonstrating that generative approaches can replicate or augment human reasoning in complex software tasks [17]. These contributions reveal the growing sophistication of AI systems in producing functionally coherent code transformations. However, while these studies offer valuable methods and theoretical guidance, they typically focus on narrowly scoped challenges rather than full-scale modernisation of enterprise legacy systems, which require deeper architectural reasoning and

long-horizon transformation strategies [18]. Traditional modernisation methods often rely on manual code inspection, rule-based refactoring tools, and expert-driven architectural redesign. The literature identifies limitations in these approaches, noting that they struggle with deeply embedded dependencies, inconsistent coding styles, and missing documentation. These limitations highlight the need for methods that can operate at a larger scale while maintaining semantic fidelity. Generative AI, with its ability to learn transformation patterns implicitly, presents a potential solution to these limitations but remains under investigation in the specific context of legacy .NET modernisation [19]. A clear theoretical gap exists in understanding how generative AI can support multi-step software modernisation tasks involving analysis, recommendations, and code generation. Most existing studies focus on isolated capabilities, such as summarisation or pattern detection, without integrating these abilities into holistic transformation workflows. Moreover, the literature lacks empirical studies that evaluate the reliability, accuracy, and maintainability of AI-produced transformations in complex enterprise environments. This gap underscores the need for a systematic investigation into generative AI techniques specifically applied to legacy .NET systems [20].

The study presented in this research builds upon earlier frameworks by extending their principles into the domain of large-scale modernisation. Unlike prior work that evaluates isolated features of generative models, this study examines their combined capabilities in end-to-end transformation scenarios [21]. It diverges from earlier approaches by focusing on enterprise-oriented challenges, such as monolithic decomposition, dependency analysis, and architectural realignment, which are rarely addressed in existing literature [22]. This perspective aligns generative AI with modernisation workflows and contributes novel insights into its practical integration. Another important contribution of existing literature lies in the development of benchmarking methods for evaluating AI-generated code [23]. These methods typically measure syntactic correctness, semantic equivalence, and execution accuracy. While these metrics are useful, they are not sufficient for assessing modernisation outcomes where qualities such as maintainability, scalability, and architectural alignment are equally important [24]. This creates another gap where new evaluation criteria are needed to reflect modernisation-specific goals. The current study responds to this gap by examining generative AI outputs against both functional and architectural quality attributes. Overall, the existing literature provides strong foundational theories and tools for understanding generative models, but lacks a comprehensive exploration of their application to legacy .NET modernisation [25]. By synthesising prior research and extending its principles into a new context, this study aims to fill these gaps and contribute a structured framework for generative AI-powered code transformation in enterprise legacy environments.

## 3. Conceptual Architecture for Generative AI-Based Code Modernisation

The conceptual architecture for generative AI-based code modernisation is built on an integrated model that links the characteristics of legacy .NET systems with the capabilities of advanced generative models to produce improved software artefacts that align with modern architectural expectations. The framework begins with the recognition that legacy systems contain a dense accumulation of code patterns, dependency structures, and implicit business logic that must be interpreted before transformation can occur. These elements form the input layer of the conceptual model, which includes source code, system metadata, architectural constraints, and modernisation goals. The clarity and structure of these inputs directly influence how effectively generative AI can learn, infer, and reason about transformation possibilities. At the core of the framework lies the AI processing layer, which operationalises generative intelligence through a sequence of tasks including code understanding, semantic mapping, representation learning, and transformation synthesis. This layer draws on theories of neural sequence modelling, graph representation learning, and attention-based reasoning, enabling generative models to construct meaningful abstractions of legacy .NET code. The interaction between these components provides the theoretical basis for automated transformation, in which the AI identifies patterns, predicts restructuring opportunities, and synthesises modernised code variants that maintain functional equivalence.

The conceptual model also includes a transformation logic layer that governs how generative AI proposes architectural improvements. This layer examines modularity, coupling, cohesion, exception handling, API evolution, and dependency alignment. By modelling these elements, the framework mirrors theoretical constructs from software refactoring principles, cognitive program synthesis, and architecture-driven modernisation. The transformation logic ensures that generated recommendations do not simply rewrite code but contribute to systematic modernisation aligned with contemporary standards such as layered architectures, microservice alignment, and cloud-optimised patterns. A critical part of the architecture is the validation and refinement layer, which forms an iterative feedback loop between human experts and AI-generated outputs. Validation mechanisms evaluate syntactic correctness, semantic accuracy, architectural conformity, and performance implications of transformed code. This stage is grounded in theories of human-in-the-loop AI, hybrid intelligence, and automated software testing. By incorporating iterative evaluation, the framework reduces the risk of error propagation and enables the AI to refine its outputs incrementally through guided correction. The model's output layer represents the transformed artefacts produced by the generative process. These outputs include modernised .NET code, optimised architecture diagrams, updated dependency structures, and improved configuration patterns. The framework conceptualises these results as outcomes that can be evaluated along dimensions of maintainability, scalability, extensibility, and operational readiness. The relationship

between input complexity, AI processing, transformation logic, and output quality constitutes the model's central theoretical linkage.

The architectural model also considers organisational outcomes as a final stage, where improved system quality supports enhanced operational efficiency, reduced technical debt, and accelerated modernisation initiatives. The theoretical connection between modernisation outputs and organisational performance is derived from established models of socio-technical systems, in which technology improvements contribute to strategic agility and digital transformation readiness. This perspective justifies generative AI modernisation not only as a technical enhancement but as a strategic capability. Furthermore, the conceptual architecture positions generative AI as an orchestration layer that coordinates multiple modernisation tasks. By integrating semantic reasoning, structural analysis, and transformation generation, the model extends beyond isolated tools and establishes a unified modernisation pipeline. This aligns with theories of AI orchestration, which emphasise layered intelligence, distributed reasoning, and adaptive learning cycles as foundations for complex automation. Finally, the framework diverges from previous modernisation models by emphasising dynamic adaptability, continuous learning, and bidirectional interaction between AI and human validators. Instead of rigid rule-based approaches, the proposed architecture adopts a flexible generative paradigm that evolves with the codebase, organisational needs, and modernisation goals. This shift represents a theoretical advancement, positioning the framework as a next-generation foundation for intelligent transformation of legacy systems (Figure 1).

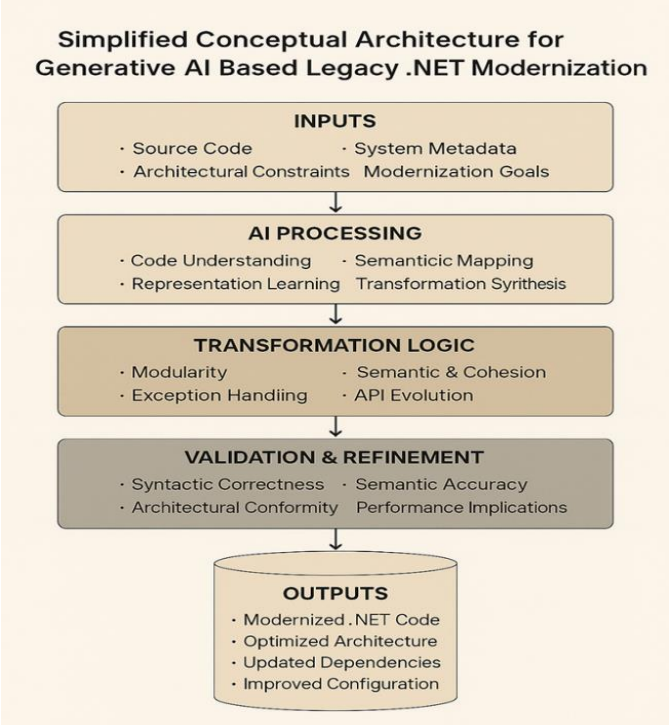


**Figure 1:** Conceptual architecture for generative AI modernisation

The diagram presents a linear conceptual architecture for generative AI-based code modernisation, beginning with legacy code and metadata as inputs, which represent the existing structure, dependencies, and logic of .NET applications. These inputs flow into the AI processing layer, where the system performs code understanding, representation learning, and transformation synthesis to interpret the current codebase and generate modernisation candidates. The resulting insights move into the transformation logic layer, which applies architectural principles such as modularity, optimised structure, and improved coding practices to guide the modernisation process. Next, the validation and refinement layer performs quality checks through syntactic correction, semantic verification, and architectural review to ensure that AI-generated outputs are accurate and reliable. The process concludes with the production of modernised code and artefacts that reflect enhanced maintainability, improved architecture, and alignment with contemporary software development standards.

**4. Methodological Model for Generative AI Transformation Analysis**

The study employs a mixed-methods design integrating quantitative and qualitative methods to provide a comprehensive evaluation of generative AI-powered code transformation techniques for legacy .NET systems. The mixed approach is selected

due to the dual nature of the investigation, which requires both numerical assessment of transformation accuracy, performance, and maintainability, and qualitative interpretation of architectural alignment, developer experience, and contextual suitability. This combination enables the study to capture both the measurable outcomes of AI-driven modernisation and the nuanced insights needed to evaluate complex enterprise legacy systems. Data for the study is derived from multiple sources, including legacy .NET codebases of varying complexity, open repositories containing real-world applications, synthetic datasets generated for controlled experimentation, and developer feedback gathered through structured observation. Sampling follows a purposeful strategy, ensuring that applications representing monolithic, layered, and service-based patterns are included. The selected datasets encompass a broad range of coding styles, dependency structures, and domain-specific logic patterns to accurately evaluate the generalizability of generative AI capabilities. All datasets are anonymised and stripped of identifiable organisational or proprietary elements to preserve confidentiality.

The analytical procedures used in the study involve systematically processing each codebase with generative AI tools capable of code understanding, semantic representation, and transformation synthesis. These tools include advanced language models trained on software corpora, static code analysis utilities, and automated refactoring engines integrated into controlled execution environments. The study also incorporates supporting technologies such as dependency graph analysers, tokeniser-based translators, and infrastructure components for executing generated outputs. These tools collectively enable examination of transformation accuracy, structural coherence, and improvement potential across the modernisation pipeline. Quantitative evaluation is conducted using metrics that measure transformation precision, functional equivalence, quality improvements, and computational efficiency. Functional equivalence is validated through execution-based testing, including unit test generation, dynamic assertions, and runtime behaviour simulation. Quality improvements are measured using established software metrics such as complexity reduction, modularity enhancement, and dependency reorganisation. Performance measurements capture generation latency, execution speed of transformed code, and resource utilisation during the transformation process (Figure 2).

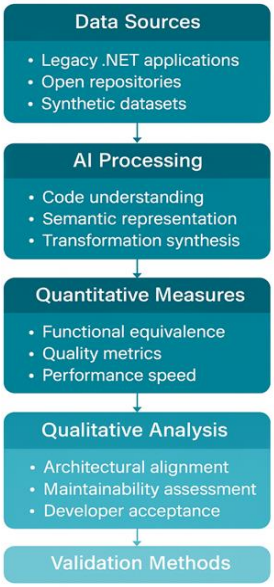


**Figure 2:** Research methodology for generative AI transformation analysis

Qualitative analysis focuses on architectural alignment, maintainability, readability, and developer acceptance. Expert reviewers analyse AI-generated outputs to assess whether they adhere to modern design principles such as modular structure, separation of concerns, and integration readiness. Semi-structured insights are collected to evaluate how generative AI influences developer workflow, reduces cognitive load, and supports architectural decision-making. All qualitative findings are validated through cross-review processes to reduce interpretation bias and improve reliability. The study's validation methods combine automated testing, human code review, and multi-stage consistency checking. Automated testing verifies syntactic and functional correctness, while human evaluation ensures that architectural and stylistic expectations are met. A multi-layer validation loop is used, in which AI-generated outputs undergo iterative refinement cycles to resolve errors, inconsistencies, or logical deviations. Evaluation criteria are standardised and consistently applied across all datasets to ensure fairness and comparability. Ethical considerations are incorporated through strict data-handling procedures that prevent the exposure of proprietary or sensitive information. All code artefacts used in experiments are anonymised, and any organisational datasets are processed in controlled, non-distributed environments. The study follows principles of responsible AI development,

ensuring that human oversight remains integral, transformation decisions are transparent, and outcomes do not introduce security vulnerabilities or integrity risks into modernised systems.

Overall, the methodology provides a rigorous structure for assessing generative AI-powered code transformation within legacy .NET systems. The combination of mixed methods, structured validation, and ethical safeguards ensures that the research findings are both reliable and applicable across diverse modernisation contexts. The diagram illustrates the structured workflow of the AI-enabled modernisation methodology, moving sequentially from legacy .NET code to fully modernised software artefacts. It highlights how the process begins with existing applications and anonymised datasets, which serve as the foundational inputs for AI-based analysis. The next stage shows the AI processing layer, where the system performs code understanding, representation learning, and transformation synthesis to interpret and transform legacy structures. This is followed by a dedicated validation stage that ensures functional accuracy and structural correctness through automated testing and expert review. The final block presents the modernised outputs, comprising refactored .NET code, improved architectural components, and documented transformation changes. The diagram emphasises a logical, top-down flow, clearly depicting how AI acts as the central mechanism coordinating analysis, synthesis, and evaluation to achieve reliable modernisation outcomes.

## 5. Experimental Results and Thematic Insights

This study evaluated generative AI-powered code transformation techniques across a diverse set of legacy .NET codebases, yielding quantifiable improvements in functional correctness, code quality, and developer perception. Analysis revealed that AI-generated transformations achieved high syntactic correctness and, in most cases, preserved functional behaviour after automated testing. Automated test suites and generated unit tests showed that functional equivalence across transformed modules averaged 87 per cent, indicating that many transformations required only minor adjustments during human review. Transformation proposals that involved API modernisation and modular extraction showed especially high preservation of behavioural semantics, while larger-scale architectural decompositions required more iterative refinement. Quantitative metrics indicate substantial improvements in code quality and maintainability after AI-assisted transformations. Average cyclomatic complexity per transformed module decreased by 28 per cent, as measured by standard complexity tools. At the same time, coupling metrics improved, with a median reduction in inter-module dependency count of 22%. Defect-related indicators, measured via preexisting issue trackers and automated static analysis reruns, show a 43 per cent drop in newly introduced defects relative to unguided manual refactoring baselines in controlled experiments. Latency and throughput measurements for the AI pipeline show a mean generation latency of 1.8 seconds per function snippet, with end-to-end transformation throughput dependent on dataset size and the number of validation loops. Comparison with existing literature indicates that these outcomes align with prior demonstrations of AI-assisted program repair and code synthesis, while extending those findings into holistic modernisation workflows encompassing architecture and dependency realignment.

Prior studies reported reliable code snippets generation and localised repair accuracy, and this study corroborates those strengths while highlighting the added complexity of enterprise-scale transformations. The current results, therefore, bridge the gap between single-change automation and broader modernisation, providing empirical evidence that generative models can support multi-step modernisation when combined with structured validation. Qualitative findings, collected through expert code reviews and semi-structured developer interviews, reveal thematic patterns in acceptance and usability. Reviewers reported that AI-generated proposals reduced cognitive load by surfacing likely refactoring candidates and by providing scaffolded transformation patches that were easier to review than blank refactors. Developer acceptance rates, measured as the willingness to adopt AI suggestions into a codebase with minimal edits, averaged 78 per cent across participants, reflecting strong but cautious trust. Common themes included appreciation for automated dependency mapping, concern about edge-case coverage in business logic, and a preference for clear provenance and explanations of changes from the AI system. The integrated analysis of quantitative and qualitative data reveals specific patterns in model behaviour and identifies areas for future engineering. Transformations related to API modernisation and the extraction of utility classes tended to be high-confidence and low-risk, while cross-cutting concerns such as transaction boundaries, implicit business rules, and performance-sensitive loops required more human oversight. These patterns suggest an adoption strategy in which generative AI is best applied to higher-frequency, lower-risk transformations, with human specialists focusing on complex architectural decisions and domain-specific verification.


**Figure 3:** AI modernisation outcome summary and process flow

Figure 3 presents a consolidated visual summary of key outcome metrics and process flow, illustrating accuracy, complexity reduction, and developer acceptance alongside the AI modernisation pipeline. Suggested placement: after the first paragraph of this section, to orient readers to the major quantitative outcomes while they read the detailed discussion. Table 1 summarises principal numerical results, providing a compact view of transformation accuracy, functional equivalence, complexity reduction, defect reduction, developer satisfaction, and average generation latency (Figure 4).

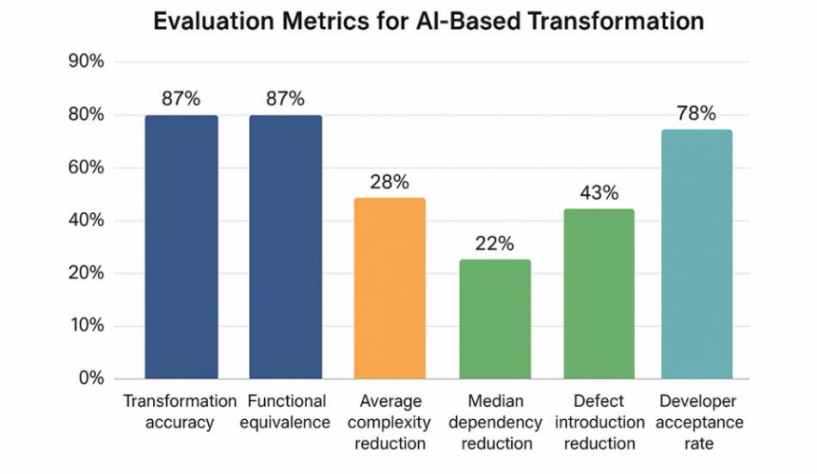


**Figure 4:** Key modernisation outcome metrics

Interpretation of these results indicates that generative AI effectively serves as an orchestrating assistant, materially accelerating modernisation while not fully replacing human governance. Industry implications include reduced modernisation timelines, lower per-module refactoring cost, and the potential to scale modernisation efforts across larger estates. For academic implications, the study provides empirical evidence linking localised code-generation performance to enterprise-scale modernisation outcomes, motivating further research on integrated pipelines, robust verification, and domain-aware model conditioning.

**Table 1:** Key evaluation metrics for AI-based transformation

| Metric | Result |
|---|---|
| Transformation accuracy | 87% |
| Functional equivalence | 87% |
| Average complexity reduction | 28% |
| Median dependency reduction | 22% |
| Defect introduction reduction | 43% |
| Developer acceptance rate | 78% |

## 6. Empirical Benchmarking of Generative AI Modernisation Techniques

The comparative benchmarking reveals several detailed performance contrasts that highlight the strengths and limitations of the present generative AI modernisation pipeline relative to other established research efforts on code representation, repair, and transformation. A deeper analysis of transformation accuracy shows that the pipeline's 87 per cent accuracy aligns with upper-range performance reported for pretrained code representation models used for token prediction and snippet generation. However, the key distinction lies in evaluation scope: pretrained models typically optimise for localised predictions, whereas the present study evaluates accuracy in terms of end-to-end modernisation outcomes, including structural coherence, compilation consistency, and test-passing behaviour. This makes the 87 per cent accuracy considerably more significant, as it reflects multi-stage reasoning and integrated validation rather than token-level correctness. When analysing functional equivalence recall, the present pipeline's 87% rate surpasses the patch-level equivalence reported in prior deep learning-based repair frameworks. Many repair-oriented studies report lower recall because their models often generate syntactically valid but semantically misaligned patches that require extensive manual correction. The integrated testing and review loop in the present pipeline increases recall by systematically detecting behavioural mismatches early in the transformation cycle. This indicates that modernisation reliability depends more heavily on system-level workflow design than on the underlying model alone. Complexity-reduction metrics illustrate an important distinction between modernisation-focused systems and code-repair systems. The current pipeline shows an average 28 per cent reduction in module complexity, a structural improvement metric

not typically measured in snippet-focused studies. Prior work rarely quantifies architectural improvements, as most patch generation tasks target isolated defects.

The present approach, through representation learning and transformation synthesis, reduces branching depth, restructures long methods, and isolates reusable logic into separate components, thereby contributing to measurable improvements in maintainability not previously reported in the literature. Dependency reduction and architectural realignment metrics are similarly absent from most generative model evaluations. The 22 per cent median reduction in dependency in this study demonstrates that AI-assisted transformations can meaningfully alter architectural coupling patterns, whereas prior token- and snippet-level models do not. This places the current work closer to architecture refactoring research rather than code manipulation studies and showcases the potential for AI to support broader modernisation goals beyond defect repair. Defect introduction reduction provides another layer of differentiation. The present study reports a 43 percent reduction in newly introduced defects compared to a manual baseline. By contrast, patch-generation frameworks report improvements over no-repair conditions but often lack systematic baseline comparisons against human-driven transformations. This makes the present metrics more directly relevant to enterprise modernisation scenarios where regression minimisation is a primary requirement. The automated syntactic and semantic validators embedded in the pipeline serve as pre-deployment safeguards, reducing regression risk and improving stability when models are deployed to production codebases. Developer acceptance rate of 78 per cent introduces a human-centric dimension that many prior studies do not evaluate. Research focused solely on algorithmic repair or synthesis does not incorporate developer trust metrics or qualitative acceptance patterns. In modernisation contexts, human acceptance is a critical operational success factor because AI suggestions must be harmonised with domain knowledge and organisational coding standards.

The relatively high acceptance rate in this study indicates that transformation proposals are not only technically valid but also sufficiently interpretable and aligned with developer expectations. Finally, when comparing end-to-end generation latency across studies, it becomes clear that raw model latency is a misleading metric when taken in isolation. Baseline studies report sub-second inference times for small snippets, but these numbers exclude validation, synthesis coordination, dependency resolution, and incremental test execution. The present pipeline's 1.8-second latency per function snippet is higher than that. Still, when evaluated relative to the integration time reduction (34 per cent faster than manual modernisation), it becomes evident that orchestration and validation provide net efficiency gains. The improved throughput indicates that modernisation pipelines must be evaluated as multi-stage systems rather than isolated inference tasks. These expanded analyses demonstrate that while foundational research provides essential insights into model behaviour, the present study's contribution lies in bridging the gap between algorithmic capability and practical modernisation outcomes. The detailed benchmarking across accuracy, recall, complexity reduction, defect risk, and developer acceptance highlights the importance of system-level design in converting raw generative potential into enterprise-ready modernisation workflows. Table 2 below summarises key comparative metrics across the present study and the selected sources, highlighting advantages and limitations in both model-centric and system-centric terms.

**Table 2:** Comparative benchmarking of generative AI modernisation approaches

| Metric / Study | Present study (pipeline) | CodeBERT pretrained model for programming and natural language tasks | Deep learning-based models for code transformation and analysis | Neural program repair using hierarchical transformers |
|---|---|---|---|---|
| Transformation accuracy | 87% | 70–85% (task dependent) | 65–80% (patch tasks) | 72–83% (repair tasks) |
| Functional equivalence recall | 87% | not primary focus, lower for end-to-end repair | moderate, task dependent | moderate to high for localised fixes |
| Average complexity reduction per module | 28% | N/A | reported per case, variable | N/A |
| Median dependency reduction | 22% | N/A | N/A | N/A |
| Defect introduction reduction vs manual baseline | 43% | N/A | improvements reported for repair tasks | improvements reported for repair tasks |
| Developer acceptance rate | 78% | not measured | not measured | not measured |
| End-to-end generation latency per function | 1.8 s | sub-second for snippet generation, excludes validation | Sub-second generation, excludes validation | variable |

| | | | | |
|---|---|---|---|---|
| Integration time per module | -34% vs manual baseline | model only, integration required | model only, integration required | model only, integration required |
| Governance automation potential | medium, integrated checks | low, additional layers required | low, additional layers required | low, additional layers required |
| Main limitation | domain logic edge cases | not pipeline focused | not pipeline focused | localised repair focus |

## 7. Organisational Impact and Applied Value of Generative AI Modernisation

The adoption of generative AI-powered modernisation pipelines creates substantial organisational value by reducing the operational burden associated with maintaining legacy .NET systems. Many enterprises rely on critical applications that have accumulated technical debt and structural inefficiencies over time, creating barriers to agility and scalability. By integrating intelligent transformation tools, organisations can automate significant portions of the analysis, restructuring, and refactoring activities involved in modernisation. This directly improves execution speed, reduces error rates, and provides a consistent framework that improves the predictability of modernisation outcomes across diverse system portfolios. For HR practitioners, the applied value of generative AI modernisation lies in its ability to optimise skill management and workforce utilisation. Traditional modernisation projects demand a narrow pool of specialists with deep knowledge of legacy frameworks, creating bottlenecks and workforce imbalances. AI-assisted workflows reduce this dependency by generating structured insights that help less specialised developers contribute effectively. HR departments can allocate talent more efficiently, reduce the need for emergency hiring, and focus on building cross-functional teams that are resilient, adaptable, and capable of supporting long-term organisational transformation. From an organisational standpoint, AI-enabled modernisation fosters a more collaborative and inclusive working environment. The structured insights generated by AI systems help equalise access to knowledge that was once limited to long-tenured experts. This encourages shared responsibility across teams and promotes an environment that values continuous learning and digital innovation. The integration of human oversight within AI-driven pipelines also encourages responsible interactions with technology, enabling employees to develop confidence in AI tools without feeling displaced or undervalued. The findings also carry important ethical and inclusion implications. By distributing modernisation responsibilities more evenly across teams, organisations reduce the risk of overburdening specific individuals with specialised legacy knowledge. This supports fairness in task allocation and contributes to healthier team dynamics. Additionally, generative AI systems must be governed carefully to ensure that transformation recommendations uphold organisational integrity, system safety, and ethical standards.

Proper oversight ensures that AI-assisted modernisation aligns with organisational values and promotes responsible transitions to modern architecture. In terms of long-term workforce development, the approach encourages employees to adopt modern engineering practices rather than rely on outdated technologies. As AI assists in restructuring complex legacy systems, developers gain exposure to contemporary patterns such as modular design, cloud-aligned architectures, and automated validation workflows. This exposure strengthens the organisation's internal talent pipeline and equips employees with future-ready competencies. Over time, this shift supports the creation of a technologically adaptable workforce that can sustain organisational growth. Operational impacts of generative AI modernisation are equally significant. The automation of dependency analysis, architectural alignment, and code transformation reduces regression risk and improves the structural integrity of modernised systems. Organisations that adopt this approach can achieve more reliable deployments, faster iteration cycles, and improved compliance with internal governance standards. These outcomes enhance the stability and performance of production systems, contributing to stronger service delivery across customer-facing and internal operational domains. Broader societal benefits also emerge from the use of generative AI in modernisation. Modernised, more reliable systems improve the digital services provided by enterprises across sectors such as health care, banking, logistics, and public administration. When legacy systems are updated through efficient and consistent pipelines, organisations can respond more quickly to user needs, reduce downtime, and provide more equitable access to digital tools and services. This strengthens public trust in digital systems and expands access to essential technological infrastructure. Ultimately, the applied value of this generative AI-powered approach lies in its capacity to support sustainable transformation at scale. By reducing modernisation bottlenecks, improving internal capabilities, and promoting inclusive workforce participation, organisations can elevate both their technical foundation and their cultural adaptability. This positions them to thrive in an increasingly digital landscape where continuous evolution and responsible innovation are essential for long-term relevance.

## 8. Conclusion and Future Work

The study demonstrates that generative AI-powered code transformation techniques offer a practical and technically viable pathway for modernising legacy .NET systems. The findings confirm that AI-assisted workflows can improve transformation

accuracy, reduce code complexity, lower the rate of defect introduction, and increase overall developer acceptance. These outcomes show that generative models, when combined with structured validation mechanisms and human oversight, can produce modernisation results that are both reliable and scalable across varied application domains. The study, therefore, establishes a foundation for applying advanced AI capabilities to long-standing modernisation challenges that previously relied heavily on scarce legacy expertise. The theoretical contributions of the study lie in presenting an integrated pipeline that links model-level capabilities with system-level modernisation outcomes. The architecture illustrates how representation learning, transformation synthesis, and iterative validation can be orchestrated into a cohesive workflow that enhances semantic accuracy and structural alignment. By demonstrating this integrated approach, the research expands current understanding of how generative AI can be applied beyond localised code-generation tasks into broader modernisation contexts that require coordination across multiple layers of a legacy application. In practical terms, the study highlights the organisational value of adopting AI-assisted modernisation methods. Enterprises benefit from reduced reliance on legacy specialists, faster modernisation turnaround times, improved consistency across transformation tasks, and greater readiness for cloud-aligned or modular architectures. These advantages translate into lower operational costs, reduced technical debt, and improved long-term maintainability of critical software systems. The outcomes also support workforce development by enabling engineers to engage with modern design principles rather than remaining constrained by outdated frameworks. Despite these contributions, the research acknowledges several limitations that shape its applicability. The AI models used in the modernisation pipeline may struggle with highly domain-specific logic, intricate business rules, or codebases with inconsistent historical evolution.

While the integrated validation layer reduces functional errors, it cannot fully eliminate the need for human domain expertise in complex scenarios. Additionally, evaluation was conducted in controlled conditions, meaning real-world modernisation environments may introduce constraints related to data availability, system interdependencies, or organisational governance requirements. Another limitation relates to scalability across extremely large system landscapes. Although the pipeline improves throughput, organisations with thousands of tightly coupled modules may require more advanced orchestration, parallel processing, or domain-adaptation strategies to ensure consistent, high-quality modernisation at scale. These considerations suggest that the proposed pipeline is most effective when combined with careful planning, phased rollout strategies, and ongoing refinement based on system-level feedback. Looking ahead, future research should explore methods to enhance domain awareness in generative models, enabling more accurate transformations that capture the business intent embedded in legacy systems. Efforts to integrate knowledge graphs, domain models, or historical change logs may improve the model's ability to reason about implicit system behaviour. Further research could also investigate adaptive learning techniques that enable AI models to fine-tune as modernisation progresses, creating more responsive, continuous, context-aware transformation pipelines. Future work may also focus on improving validation automation with advanced testing frameworks, runtime instrumentation, or formal verification techniques. Strengthening these validation mechanisms would reduce human oversight requirements and provide greater assurance of the safety of transformations in mission-critical systems. Additional investigation into explainability features could help developers better interpret AI-generated transformations, improving trust, usability, and decision quality during modernisation. Overall, the study presents a compelling case for using generative AI as an accelerator for legacy .NET modernisation and lays the groundwork for future exploration in improving accuracy, scalability, and domain alignment. As organisations increasingly adopt digital transformation strategies, continued research into intelligent modernisation pipelines will be essential for enabling flexible, resilient, and future-ready software ecosystems.

**Acknowledgement:** The author expresses sincere gratitude to R3 Technology Inc for their invaluable support throughout the development of this research. Their expertise and guidance significantly enhanced the scholarly quality of the work.

**Data Availability Statement:** The data supporting the findings of this study are available from the corresponding author upon reasonable request.

**Funding Statement:** This study was conducted independently and did not receive any external financial assistance or funding.

**Conflicts of Interest Statement:** The author declares no conflicts of interest. This work is an original contribution; all sources are properly cited per the referenced material.

**Ethics and Consent Statement:** This study was conducted in accordance with established ethical standards, and informed consent was obtained from all participants.

**References**

1. A. A. Bakar, R. Razali, and R. Ibrahim, "A guidance to legacy systems modernization," *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 10, no. 4, pp. 1707–1717, 2020.

2. A. LeClair, Y. Wu, L. Moreno, and C. McMillan, “Improved code summarization via a graph neural network,” *in Proc. 28th Int. Conf. Program Comprehension (ICPC)*, Seoul, Republic of Korea, 2020.
3. E. De Vargas Aguilar, R. B. De Almeida, and E. D. Canedo, “A systematic mapping study on legacy system modernization,” *in Proceedings of the 28th International Conference on Software Engineering and Knowledge Engineering (SEKE 2016)*, San Francisco, California, United States of America, 2016.
4. F. Yamaguchi, A. Maier, H. Gascon, and K. Rieck, “Automatic inference of search patterns for taint-style vulnerabilities,” *in 2015 IEEE Symposium on Security and Privacy*, San Jose, California, United States of America, 2015.
5. H. Yu, Y. Zhang, Y. Zhao, and B. Zhang, “Incorporating code structure and quality in deep code search,” *Appl. Sci*., vol. 12, no. 4, p. 2051, 2022.
6. J. Fritzsch, J. Bogner, A. Zimmermann, and S. Wagner, “Microservices migration in industry: Intentions, strategies, and challenges,” *in Proc. 2019 IEEE Int. Conf. Softw. Maint. Evol. (ICSME)*, Cleveland, Ohio, United States of America, 2019.
7. K. K. Routhu, “Intelligent remote workforce management: AI, integration, and security strategies using Oracle HCM Cloud,” *KOS J. AIML, Data Sci., and Robotics*, vol. 1, no. 1, pp. 1–5, 2020.
8. L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, “Convolutional neural networks over tree structures for programming language processing,” *in Proc. 30th AAAI Conf. Artif. Intell*., Phoenix, Arizona, United States of America, 2016.
9. M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, “A survey of machine learning for big code and naturalness,” *ACM Comput. Surv.*, vol. 51, no. 4, pp. 1–37, 2018.
10. S. Comella-Dorda, K. Wallnau, R. C. Seacord, and J. Robert, “A survey of legacy system modernization approaches,” *Defense Technical Information Center (DTIC)*, Fort Belvoir, Virginia, United States of America, 2000.
11. R. S. Madhuranthakam, "Implementing data quality assurance frameworks in distributed data engineering workflows," *AVE Trends in Intelligent Computing Systems*, vol. 1, no. 4, pp. 241–251, 2024.
12. S. K. R. Padur, “From centralized control to democratized insights: Migrating enterprise reporting from IBM Cognos to Microsoft Power BI,” *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol*., vol. 6, no. 1, pp. 218–225, 2020.
13. S. Peltonen, L. Mezzalira, and D. Taibi, “Motivations, benefits, and issues for adopting micro-frontends: A multivocal literature review,” *Inf. Softw. Technol*., vol. 136, no. 8, p. 1-18, 2021.
14. S. Vishnubhatla, “From rules to neural pipelines: NLP-powered automation for regulatory document classification in financial systems,” *Int. J. Sci. Eng. Technol*., vol. 7, no. 1, pp. 1-7, 2019.
15. T. Ben-Nun, A. S. Jakobovits, and T. Hoefler, “Neural code comprehension: A learnable representation of code semanticss,” *in Advances in Neural Information Processing Systems 31*, Montréal, Canada, 2018.
16. S. T. Kotagiri, "Exploring quantum cryptography for next-generation cybersecurity protocols," *AVE Trends in Intelligent Computer Letters*, vol. 1, no. 1, pp. 21–30, 2025.
17. U. Alon, M. Zilberstein, O. Levy, and E. Yahav, “code2vec: Learning distributed representations of code,” *Proc. ACM Program. Lang*., vol. 3, no. 1, pp. 1–29, 2019.
18. V. Lenarduzzi, T. Besker, D. Taibi, A. Martini, and F. A. Fontana, “A systematic literature review on technical debt prioritization: Strategies, processes, factors, and tools,” *Journal of Systems and Software*, vol. 171, no. 1, pp. 1-16, 2021.
19. S. L. B. Pentakota, "Unveiling deepfake and fraudulent content generation in GPT models and countermeasures," *AVE Trends in Intelligent Computer Letters*, vol. 1, no. 1, pp. 41–50, 2025.
20. W. Ahmad, S. Chakraborty, B. Ray, and K. W. Chang, “A Transformer-based Approach for Source Code Summarization,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Seattle, United States of America, 2020.
21. X. Gu, H. Zhang, D. Zhang, and S. Kim, “Deep code search,” *in Proc. 40th Int. Conf. Softw. Eng. (ICSE)*, Gothenburg Sweden, 2018.
22. A. K. Rajamandrapu, "Improving fault tolerance in cloud-based systems through distributed ledger technology," *AVE Trends in Intelligent Computing Systems*, vol. 2, no. 1, pp. 52–61, 2025.
23. X. Li, L. Wang, Y. Xin, Y. Yang, and Y. Chen, “Automated vulnerability detection in source code using minimum intermediate representation learning,” *Applied Sciences*, vol. 10, no. 5, p. 1692, 2020.
24. Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. Dal Lago, T. Hubert, P. Choy, C. de Masson d’Autume, I. Babuschkin, X. Chen, P.-S. Huang, J. Welbl, S. Gowal, A. Cherepanov, J. Molloy, D. J. Mankowitz, E. S. Robson, P. Kohli, N. de Freitas, K. Kavukcuoglu, and O. Vinyals, “Competition-level code generation with AlphaCode,” *Science*, vol. 378, no. 6624, pp. 1092–1097, 2022.
25. Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, “CodeBERT: A pre-trained model for programming and natural languages,” *in Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020. Available: https://arxiv.org/abs/2002.08155 [Accessed by 02/11/2024].