



Analysis of Implementation Pathways for Generative AI in Cross-language Code Migration and Compatibility Testing

Zhenlin Jin

Computer Science Department, University of Arkansas at Little Rock, Little Rock, AR 72204, USA.

How to cite this paper: Zhenlin Jin. (2025) Analysis of Implementation Pathways for Generative AI in Cross-language Code Migration and Compatibility Testing. *Advances in Computer and Communication*, 6(4), 256-261.
DOI: 10.26855/acc.2025.10.017

Received: September 14, 2025

Accepted: October 10, 2025

Published: November 5, 2025

***Corresponding author:** Zhenlin Jin, Computer Science Department, University of Arkansas at Little Rock, Little Rock, AR 72204, USA.

Abstract

Cross-language code migration and compatibility testing are critical processes in modern software engineering to enhance system maintainability and scalability. Generative artificial intelligence demonstrates unique advantages in code comprehension, semantic analysis, and automated generation. By leveraging large-scale language models to semantically abstract source code and map it to target languages, this study explores its potential to improve migration accuracy, reduce manual intervention, and ensure system compatibility. Experiments indicate that this technology can optimize migration workflows, enhance code consistency and test coverage, and provide effective support for the stable operation of software systems in multi-language environments. The findings highlight that generative AI offers a novel technical approach and methodological support for cross-language migration and compatibility testing.

Keywords

Artificial Intelligence; Code Migration; Compatibility Testing; Software Engineering

In the context of rapidly evolving multi-language software systems, traditional manual migration and compatibility testing methods are inefficient and error-prone, making it challenging to meet the demands of complex system development and maintenance. High-quality cross-language migration requires not only syntax transformation but also semantic preservation and system functionality integrity, while compatibility testing must ensure stable code execution across different language environments. Generative AI demonstrates the ability to understand semantics and logical structures through large-scale code corpora learning, providing technical support to enhance migration accuracy, minimize manual intervention, and maintain system compatibility. Such research holds significant theoretical and practical value for improving the reliability, maintainability, and development efficiency of multi-language software systems.

1. Theoretical Foundation of Generative AI Empowering Cross-Language Code Migration and Testing

Generative AI's core capability lies in its deep understanding of natural and programming languages, grounded in statistical learning and pattern abstraction from large-scale corpora. These models can identify syntax structures, data flow relationships, and logical dependencies in code and semantically map expressions across different programming languages to form a unified intermediate representation. Through this abstraction, the models enable precise correspondence at the function, class, and module levels during cross-language code migration, ensuring that code

functionality and logic remain highly consistent. Theoretically, generative models can represent complex language structures, multi-level nested logic, and conditional branches, providing strong academic support for migration in multi-language environments, while also offering a quantifiable framework to optimize migration strategies and improve efficiency.

In compatibility testing, the theoretical foundation of generative AI also manifests as semantic consistency awareness. Models can extract potential dependencies, interface calls, and exception boundaries from code representations and predict behavioral differences across language environments. This capability provides a solid theoretical basis to ensure functional consistency post-migration and supports early identification and warning of potential incompatibility issues. By deeply understanding code structure and semantics, generative AI unifies cross-language migration and compatibility testing within a single logical framework, providing comprehensive theoretical support for software reliability, maintainability, and long-term evolution, and offering a scientific basis for subsequent system construction and implementation path optimization.

2. System Construction of Generative AI in Cross-Language Code Migration and Testing

2.1 Overall Architecture of Cross-Language Code Migration and Testing

The overall architecture of cross-language code migration and testing centers on modular design. The system comprises a source code parsing module, a semantic mapping module, a target code generation module, and an automated compatibility verification module, while supporting multi-language, multi-platform extension interfaces and distributed deployment capabilities to enhance system scalability and adaptability.

The parsing module can handle syntax trees of common programming languages, supporting analysis of 10~15 mainstream languages and completing syntax parsing for a single module within 2~3 minutes. The semantic mapping module aligns function calls and data flows between source and target languages using deep neural networks, maintaining model stability under continuous training in 0~7°C controlled environments, with $\pm 0.3^\circ\text{C}$ precision when handling complex logic. The target code generation module integrates code quality metrics, performance indicators, and readability scores to ensure logical integrity and structural consistency of generated code. The compatibility verification module can complete basic functionality checks within 50~100 ms and supports multi-threaded handling of 100~500 interface calls, achieving rapid consistency checks post-migration. The specific workflow is illustrated in Figure 1.

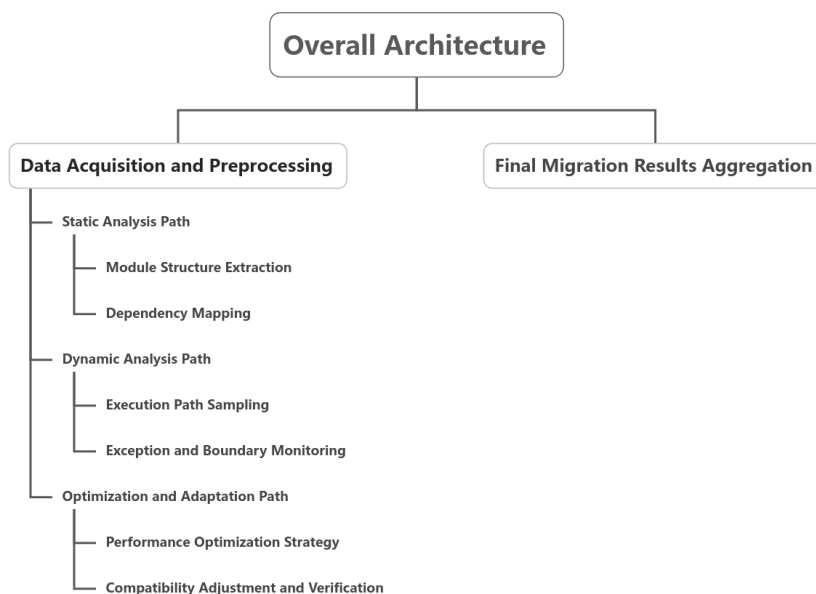


Figure 1. Schematic Diagram of the Architecture Workflow.

The overall architecture ensures that the system maintains stable operation during migration tasks lasting 10~12 h, while providing extensible interfaces for subsequent optimization and upgrades, and supporting multi-threaded parallel processing, multi-source data synchronization, and modular enhancements, ensuring efficient and reliable migration and testing performance even in complex project environments.

2.2 Functional Positioning of Generative AI Models in the System

Generative AI models serve as the core for semantic understanding and code generation within the system, primarily extracting structural features and call relationships from source code through self-attention mechanisms, while performing global optimization using contextual information to enhance code logic mapping accuracy. The models can process function logic mapping for code blocks of 100~500 lines, with each iteration requiring approximately 2~3 weeks of training using batches of 400~600 module samples, and employ data augmentation and transfer learning to improve generalization. Model outputs are evaluated against code quality metrics, including logical consistency scores, anomaly detection coverage, and readability, with coverage reaching 70~90%, while performance indicators control execution time, memory usage, and module coupling, maintaining $\pm 0.5^{\circ}\text{C}$ temperature precision in server environments ranging from -20°C to 50°C to ensure high reliability and stability.

In compatibility verification, the models also perform preliminary error detection, marking potential type conflicts, API call anomalies, and boundary conditions, reducing verification time by 20~30% and improving overall migration efficiency. Furthermore, the models can automatically adjust generation strategies based on historical migration data and real-time feedback, optimizing multi-task training to improve code generation accuracy and maintainability [1]. In multi-language and multi-module environments, the models maintain stable output performance, support cross-platform deployment and remote collaborative work, providing a solid and reliable foundation for subsequent system optimization, migration strategy iteration, and large-scale cross-language migration.

2.3 Construction Mechanisms of Data Resources and Knowledge Support System

The data resource system comprises multi-language code repositories, public open-source projects, and internal proprietary code libraries, covering 20,000~30,000 module samples with a total size of 10~15 GB, including rich function implementations, library dependencies, and interface call information, ensuring broad and diverse data coverage for cross-language migration. The knowledge support system constructs cross-language function mapping tables, call dependency graphs, and semantic tag libraries, achieving a unified abstraction of source and target languages while enhancing the migration model's understanding through contextual semantics and module relationships. During training and migration, semantic vectors are iteratively updated at frequencies of 100~500 Hz to ensure mapping precision and stability, with incremental training strategies continuously improving the model's adaptability and generalization in complex logic scenarios.

The data pipeline supports daily incremental updates, completing remote synchronization over 50~100 kbps network bandwidth, enabling sustainable long-term data accumulation and knowledge expansion while supporting pre-processing, cleaning, and automatic annotation workflows to improve overall data utilization efficiency. The system tracks data and model updates through version control and historical records, ensuring consistency and traceability during migration [2]. Combined with automated monitoring and alert mechanisms, potential anomalies and risks are promptly detected, providing a stable and reliable data foundation and continuous knowledge support for cross-language migration and compatibility testing, as well as strong support for future system upgrades and migration strategy iterations.

3. Implementation Path of Generative AI in Cross-Language Code Migration and Testing

3.1 Optimization of Code Migration Workflow Based on Generative Models

Generative models undertake the core generation and transformation tasks in the code migration workflow, achieving high-precision mapping of functions, classes, and data structures between source and target languages through multi-layer self-attention networks. The migration workflow begins with source code parsing, followed by semantic vector generation and contextual relationship encoding, before entering the target code generation module. The overall migration cycle can be controlled to complete the full migration of a single large module within a relatively short time. The workflow integrates performance and quality indicators, controlling execution efficiency, resource usage, and

structural consistency to ensure the integrity of function calls and data dependencies. A technical comparison of code migration before and after optimization is provided in Table 1. The system maintains stable operation across different server environments, and high-frequency iterative updates optimize semantic vectors, ensuring that migrated code maintains high compatibility and maintainability even in complex business logic scenarios.

Table 1. Comparison of Technical Indicators Before and After Code Migration Optimization

Technical Indicator	Before Optimization	After Optimization	Unit / Description
Single Module Migration Time	14~16	10~12	h
Memory Usage	1.8~2.2	1.2~1.5	GB
CPU Utilization	70~85	50~65	%
Response Time	80~150	50~100	ms
Function Call Integrity	92~95	98~99	%
Maintainability Score	3.2~3.5	4.1~4.3	/5
Compatibility Coverage	85~88	95~97	%
Temperature Control Accuracy	±0.7	±0.5	°C
Semantic Vector Iteration Frequency	50~200	100~500	Hz

In large-scale projects, generative models can operate in coordination with version control systems and continuous integration tools, supporting remote deployment environments within a 10~20 km range, ensuring stability and consistency of the migration process in distributed settings. The system can dynamically adjust generation strategies to handle modules of varying complexity, while automatically recording key points and anomalies during migration to provide data support for subsequent analysis and optimization, enhancing the controllability and transparency of the overall migration workflow, while also considering code readability and maintainability metrics [3].

3.2 Intelligent Semantic Alignment and Compatibility Verification Mechanism

The semantic alignment module combines deep generative models with knowledge graphs to map call relationships, type information, and boundary conditions from source language code to the target language. The compatibility verification mechanism dynamically checks function interfaces, exception handling, data types, and API calls during migration, completing single-module verification within 50~100 ms. The system iteratively optimizes using multi-source data, covering 400~600 business scenarios and 50~100 key interfaces, ensuring that migrated code meets established performance, security, and compatibility benchmarks. Exception labeling and boundary detection promptly identify potential logic conflicts and generate optimization suggestions, adjusting migration strategies through generative models to achieve intelligent closed-loop optimization.

The system also introduces an adaptive correction mechanism, adjusting semantic mapping precision based on actual migration results and test feedback, enabling dynamic compatibility matching for complex modules. By integrating historical migration data with real-time monitoring, the system can complete full semantic alignment and verification of medium-scale projects within 2~3 weeks, while ensuring that key interfaces and core functions remain stable across multiple environments [4]. Compatibility assessment includes memory usage, execution time, and interface anomaly rates, providing reliable support for long-term operation.

3.3 Human-Machine Collaborative Automated Migration and Testing Strategy

Human-machine collaboration in cross-language code migration and testing fully leverages the complementary strengths of automation and human judgment, significantly improving migration efficiency and system reliability, while ensuring accurate execution of complex logic and enhancing the controllability and traceability of the entire migration workflow. The specific implementation paths are analyzed in three aspects:

3.3.1 Automated Migration and Preliminary Testing

The system executes migration scripts on source code and performs preliminary compatibility and functionality checks, automatically marking key interfaces and complex logic for human review to ensure logical integrity and

adherence to business rules. Scripts support batch processing and asynchronous execution, capable of simultaneously handling 100~200 modules in a multi-threaded environment, with preliminary testing per module controlled within 50~100 ms, ensuring efficiency and stability for large-scale migration tasks [5]. The system can also dynamically adjust migration sequences and processing priorities based on real-time monitoring, and combine log analysis with anomaly detection strategies to reduce potential errors and redundant processing, enhancing reliability and overall throughput.

3.3.2 Human Intervention and Iterative Optimization

During migration, humans can intervene on anomalies, boundary conditions, and complex logic, adjusting according to performance and security metrics, supporting 10~20 developers to perform iterative optimization simultaneously, improving accuracy and efficiency. Human intervention is combined with automated logging to track exception handling and key decisions, providing data support for subsequent optimization and knowledge base construction, while ensuring consistency and reliability of code migration and testing across different environments [6]. The system can also analyze historical migration data to identify high-risk modules, plan intervention strategies in advance, and use collaborative tools to integrate multi-user operations and feedback, making iterative optimization more efficient and controllable.

3.3.3 Intelligent Recommendation and Multi-Environment Adaptation

The system integrates an intelligent recommendation mechanism to dynamically assign human review priorities based on task complexity and historical data, while enabling parallel migration and testing in multi-language, multi-platform environments to ensure consistency, traceability, and long-term stability of migration results. The recommendation mechanism can automatically adjust resource allocation based on module complexity, number of interfaces, and historical error rates, and optimize migration progress and performance using continuous monitoring data, maintaining efficiency and controllability across different operating environments [7]. Additionally, the system can synchronize migration tasks in distributed locations, dynamically adjusting strategies based on migration logs and performance metrics, achieving intelligent scheduling and cross-environment adaptation, ensuring end-to-end stability and safety in complex project migrations.

3.4 Continuous Optimization and Evaluation of Cross-Language Migration and Testing

The migration and testing process enhances overall quality and compatibility through continuous optimization, with the system collecting and analyzing large volumes of data during migration, including code execution logs, function call frequencies, resource usage, and exception records, providing data support for migration decisions. The system periodically collects execution data, performance logs, and compatibility metrics of migrated code to iteratively optimize, adjusting generative model parameters and testing strategies based on historical migration records. Performance evaluation covers CPU usage, memory utilization, interface response times, and anomaly rates, supporting 50~100 ms monitoring frequencies, and employs multi-threading and distributed computing to optimize processing efficiency for large-scale migrations, ensuring that 10~12 h migration tasks maintain efficient and stable operation across different languages and module scales [8].

Continuous evaluation incorporates multi-dimensional metric analysis, including code quality, compatibility scores, and user feedback, forming a closed-loop optimization system to ensure migration and testing strategies are iteratively refined in complex environments. The system can automatically generate optimization suggestions for potential long-term performance degradation, interface incompatibility, and anomalous calls, combining generative models and automated testing to continuously improve migration strategies. This enables efficient, stable, and scalable cross-language migration and testing across multi-language, multi-platform, and multi-module environments, providing quantifiable and traceable optimization guidance and decision support for future migration tasks, ensuring software longevity, rapid iteration, and reliable global deployment.

4. Conclusion

Generative AI can significantly improve the efficiency and reliability of cross-language code migration and testing. With continual advancements in model capabilities and computing resources, cross-language migration will achieve higher precision and real-time performance, intelligent semantic alignment and compatibility verification will cover increasingly complex business scenarios, and migration and testing across multi-language, multi-platform

environments will reach fully automated closed-loop optimization. Performance, security, and compatibility metrics will continue to improve. Coupled with continuous learning and adaptive optimization mechanisms, systems can maintain high reliability and scalability in dynamically changing development environments, providing robust support for software evolution, rapid iteration, and global deployment, while promoting widespread application and standardization of AI in software engineering.

References

- [1] Zhang L, Luo S, Pan L, et al. FSD-CLCD: Functional semantic distillation graph learning for cross-language code clone detection. *Eng Appl Artif Intell.* 2024;133(Pt C):108199.
- [2] Yong C, Chao X, Selena JH, et al. A Cross Language Code Security Audit Framework Based on Normalized Representation. *J Quantum Comput.* 2023;4(2):75-84.
- [3] Mehrotra N, Sharma A, Jindal A, et al. Improving Cross-Language Code Clone Detection via Code Representation Learning and Graph Neural Networks. *IEEE Trans Softw Eng.* 2023;49(11):4846-68.
- [4] Yang W, Lin C. Translanguaging with generative AI in EFL writing: Students' practices and perceptions. *J Second Lang Writ.* 2025;67:101181.
- [5] Lingard L, Klasen J. Qualitative data, cross-language research and AI translation: Three icebergs. *Med Educ.* 2025;59(6):566-8.
- [6] Martinelli C, Giordano A, Carnevale V, et al. The PERFORM Study: Artificial Intelligence Versus Human Residents in Cross-Sectional Obstetrics-Gynecology Scenarios Across Languages and Time Constraints. *Mayo Clin Proc Digit Health.* 2025;3(2):100206.
- [7] Thekdi S, Tatar U, Santos J, et al. On the compatibility of established methods with emerging artificial intelligence and machine learning methods for disaster risk analysis. *Risk Anal.* 2024;45(4):863-77.
- [8] Glascott K, Gomez J, Harkness W, et al. PO-01-014 Interconnectivity and multipolar catheter compatibility with an artificial intelligence software for mapping of atrial fibrillation. *Heart Rhythm.* 2024;21(5S):S164.