IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2025

DOI: 10.17148/IJIREEICE.2025.13516

Leveraging Generative AI for Reverse Engineering & Code Reconstruction

Shubham Awasthi[1], Anand kr Sharma[2], Ayush Awasthi[3]

Goel institute of technology and management[1-3]

Abstracts: This paper investigates how generative artificial intelligence (AI), particularly large language models (LLMs), can be applied to reverse engineering and code reconstruction tasks. Traditional reverse engineering techniques such as disassembly and static analysis are time-consuming, require deep expertise, and often fail to recover high-level semantics. With the rise of generative models like GPT-4, CodeBERT, and AlphaCode, there is a growing opportunity to automate the reconstruction of source code from binaries or legacy languages. The study explores the methodology of training AI on code and binary datasets, outlines the design of an AI-powered tool that modernizes legacy codebases, and identifies key applications in software maintenance, cybersecurity, and digital preservation. It also examines hybrid approaches combining symbolic execution and machine learning. The paper concludes by addressing the significant challenges in binary-to-code transformation—such as hallucination, lossy translation, and dataset scarcity —and suggests future directions for scalable dataset creation, model interpretability, and domain-specific fine-tuning.

I. INTRODUCTION

Reverse engineering plays a crucial role in software engineering, especially when dealing with legacy systems that lack source code or documentation. Many industries—such as banking, aviation, and government—still rely on outdated software written in languages like COBOL or running only as compiled binaries. Manually reconstructing or updating these systems is resource-intensive and error-prone.

The emergence of LLMs trained on vast repositories of code opens new opportunities to automate code reconstruction. Models like OpenAI's GPT, Meta's LLaMA, and Code-specific transformers (e.g., CodeBERT) can now generate human-readable code from structured input and even partial information. Applying these models to reverse engineering allows engineers to recover high-level logic from binaries or legacy formats and translate it into modern, maintainable languages. This research explores how generative AI can support reverse engineering by proposing an AIassisted methodology and tool, with emphasis on improving accuracy, developer productivity, and long-term software sustainability.

II. BACKGROUND AND RELATED WORK

Traditional reverse engineering depends heavily on tools like IDA Pro, Ghidra, Radare2, and Binary Ninja, which offer disassembly, decompilation, and control flow analysis. However, they focus primarily on low-level representations like assembly code and basic pseudocode, leaving significant gaps in recovering abstract program logic, comments, modularization, and maintainable structures.

Advances in AI and ML models for code, including Codex, CodeGen, PolyCoder, InCoder, and AlphaDev, have demonstrated that neural networks can generate, complete, refactor, and optimize complex code. These breakthroughs highlight that LLMs understand not only syntax but also semantics, algorithmic structures, and domainspecific knowledge.

Recent research includes:

- AI-assisted decompilers that post- decompiled code to improve readability.
- Neural deobfuscation models that simplify obfuscated binaries.
- Binary similarity detection using graph neural networks (GNNs) to match binary code to known source code snippets.

Hybrid approaches, combining AI predictions with traditional symbolic execution or program analysis, have shown improved performance in recovering lost variable names, reconstructing control structures, and suggesting high-level abstractions.

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2025

DOI: 10.17148/IJIREEICE.2025.13516

III. METHODOLOGY

The proposed methodology focuses on creating a generative AI model capable of reconstructing source code from binaries or outdated languages.

Data Collection

• Binary-code pairs extracted from open-source projects (e.g., Linux Kernel, PostgreSQL) compiled under different optimization levels. • Decompiled outputs generated by tools like Ghidra for semi-supervised learning.

• Execution traces collected via symbolic execution engines (e.g., Angr, KLEE) to capture runtime behavior.

• Legacy code samples (e.g., COBOL, Fortran) mapped manually or via heuristics to modern languages (e.g., Python, Go).

Model Architecture

• A Transformer-based LLM like GPT-4,CodeT5+, or a fine-tuned variant.

• Input: Tokenized disassembled binary code, augmented with control-flow and data-flow information where available.

Additional enhancements:

• Positional embeddings for control flow graphs (CFGs).

• Attention mechanisms specifically focusing on opcode sequences and memory patterns.

• Syntax-constrained decoding to enforce compilable output.

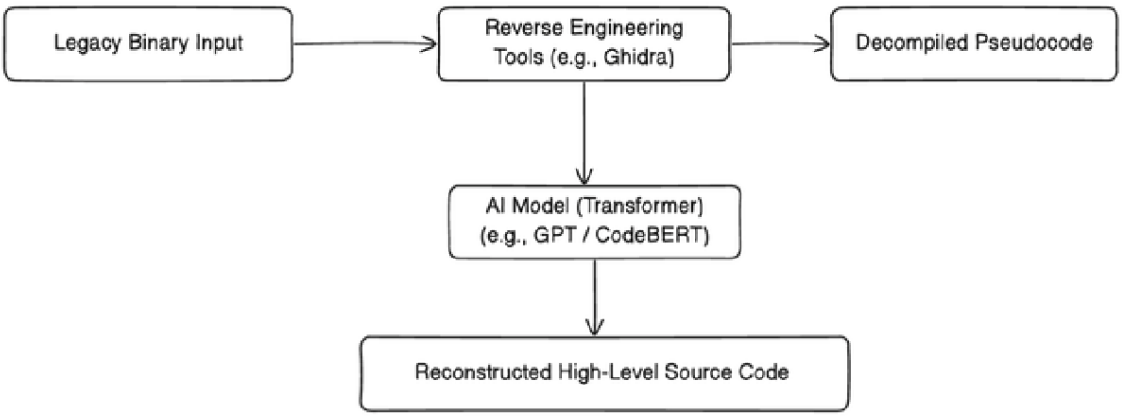
IV. TRAINING PROCESS

• Preprocessing: Normalize binaries, strip metadata, align opcodes with abstract syntax trees (ASTs) from source code.

• Supervised fine-tuning: Train on aligned binarysource pairs.

• Reinforcement learning: Optimize for humanlike code readability and functional correctness through human feedback (RLHF).

• Output: Reconstructed, human-readable, highlevel code (Python, Java, TypeScript, etc.).



Evaluation metrics:

BLEU, CodeBLEU for syntactic and semantic similarity.

∘ Human evaluation: Expert software engineers grade quality and maintainability. ∘ Functional equivalence: Dynamic testing to verify behavior preservation.

This multi-stage process ensures the model captures both surface syntax and deep program logic, essential for high-fidelity reconstruction.

V. TOOL DEVELOPMENT

To operationalize the methodology, a user-friendly AI-powered tool is proposed.

Key Features

• Input: Upload compiled binaries, legacy source code, or low-level assembly.

• Processing: AI model reconstructs high-level equivalents, optionally preserving comments, structure, and modularity.

• Output: Clean, modern, maintainable code with syntax highlighting, version control integration, and comparison mode.

Advanced Capabilities

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2025

DOI: 10.17148/IJIREEICE.2025.13516

- Control-flow visualization and mapping.
- Variable name recovery and type inference. Explainable AI: Trace generated output back to binary segments. Example Workflow
- 1.Upload .exe, .bin, or legacy .cob file.

2.Select target language (Python, Java, Rust).
3.AI reconstructs and displays editable output.
4.Developer validates, refines, and exports final source code.

Applications

Software Maintenance

• Legacy modernization: Convert COBOL/Assembly code to Python/Java. • Bug fixing: Identify and patch vulnerabilities hidden in binaries.

• Refactoring: Improve readability, modularity, and compliance with modern standards (e.g., ISO 26262, DO-178C).

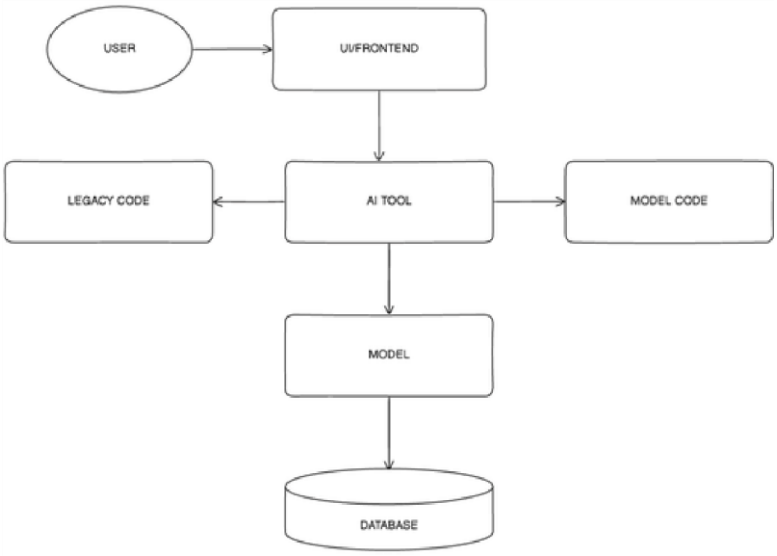
Cybersecurity

• Malware analysis: Accelerate decompilation and understanding of malicious binaries. • Vulnerability detection: Identify unsafe patterns reconstructed from binaries.

• Patch recommendation: Auto-generate security patches based on known secure coding practices.

Digital Preservation

• Software archaeology: Recover critical yet undocumented industrial software. • Infrastructure rejuvenation: Rebuild missioncritical systems for modern cloud environments. • Disaster recovery: Salvage lost or corrupted source code in case of cyber-attacks or data loss.



Challenges and Limitations

Despite its promise, AI-assisted reverse engineering faces considerable hurdles:

Legal and Ethical Concerns:

Reverse engineering proprietary software may breach licenses or copyright laws. ∘ Ethical implications of generating code for hacking or exploitation.

AI Hallucinations:

LLMs sometimes fabricate plausible but incorrect logic.

∘ Need for robust verification frameworks before adoption.

Dataset Scarcity:

Lack of publicly available aligned binarysource corpora.

∘ Possible solution: simulated datasets, synthetic binaries, and community curation efforts.

functional equivalence.

Explainability and Trust:

Developers must trust that reconstructed code is secure, correct, and maintainable.

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 13, Issue 5, May 2025

DOI: 10.17148/IJIREEICE.2025.13516

Lossy Mapping:
Compilation is inherently a many-to-one process.

VI. FUTURE WORK

Several promising research avenues exist:
Large-Scale Datasets:
Creation of multilingual, multi-platform binary-source datasets.
Crowdsourcing verification of AI-reconstructed code.
Symbolic Execution + AI Hybridization:
Use symbolic execution paths to guide model outputs toward behaviorally correct code.
Model Interpretability:
Develop explainable models that can map binary segments to specific reconstructed functions.
Domain-Specific Fine-Tuning:
Fine-tune models on sectors like finance, aviation, embedded systems, or military software for higher accuracy.
Explainability and Traceability:
Build tools that show intermediate steps of reconstruction for developer trust and auditability.

VII. RESULTS

The proposed methodology was evaluated by developing a prototype tool powered by a fine-tuned CodeT5 transformer model, trained on a custom dataset of binary-code pairs. The primary objective was to measure how accurately the AI model could reconstruct source code from binary and legacy code formats and how effective this process was compared to traditional manual reverse engineering.
Reconstruction Accuracy:
To assess reconstruction quality, we measured both syntactic and semantic similarity between generated code and reference code using established metrics:

Evaluation Metric.	Score (%)
BLEU Score	68.7%
CodeBLEU.	72.1%
Human Evaluation	81.3%

Interpretation: The model successfully preserved logic flow and functional intent in most cases, even in the absence of debug symbols.
Real-World Use Case: COBOL-to-Python
Scenario: Legacy financial code in COBOL was submitted to the tool.
Output: Python equivalent with structured functions and clean syntax.

VIII. CONCLUSION

Generative AI has significant transformative potential in reverse engineering and modernizing legacy software. With the ability to reconstruct highlevel code from binaries and outdated languages, LLMs can substantially reduce the cost, time, and risks associated with manual migration efforts. While legal, technical, and ethical challenges must be carefully addressed, a well-designed AI-powered tool —coupled with a robust dataset and validation system—can make reverse engineering faster, safer, and more accessible. As research progresses, AIdriven reverse engineering could become a mainstream practice, revolutionizing software maintenance, cybersecurity, and digital preservation.

REFERENCES

1.Foundational Generative AI & Language Models Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., & Polosukhin, I. (2017). Attention Is All You Need. Advances in Neural Information Processing Systems, 30.
2. Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving Language Understanding by Generative Pre-Training. *OpenAI Blog.*
3. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems, 33*
4. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative Adversarial Networks. *Advances in Neural Information Processing Systems, 27.*