Lab 1: Legacy Lift

AI Pipeline for Legacy Code Modernization

Aaditya Ghimire

Caleb Peterson

Darius Itam

Jordan Kemelhar

Justin Tendilla

Spencer Peloquin

Old Dominion University

CS 410

April 30, 2026

Table of Contents

1. Introduction 3
2. Legacy Lift Product Description 5
   Product Features and Capabilities 5
   Components (Hardware/Software) 6
3. Identification of Case Study 8
4. Glossary 9
5. References 10

List of Figures

Figure 1: Solution Process Flow 11

Figure 2: Work Breakdown Structure 11

List of Tables

Table 1: Feature Comparison (Legacy Lift vs. others) 6

1. Introduction

Many critical enterprise systems continue to run on legacy software written in outdated languages such as COBOL, Fortran, and Ada. These systems are deeply embedded in finance, government, healthcare, and other infrastructure sectors. For example, 95% of ATM transactions in the U.S. are processed using COBOL-based systems, and over 60% of U.S. hospitals run critical applications on legacy software such as COBOL and Fortran(Pragmatic Coders 1).

A significant problem facing modern enterprise systems is a heavy reliance on legacy code, primarily COBOL. COBOL code is much less secure and more difficult to debug than modern Object-Oriented Programming (OOP) code like Python. Furthermore, COBOL code is functionally incompatible with many of the principles of OOP, making it difficult to translate effectively. The current solution is to hire a dedicated “legacy programmer” familiar with COBOL to manually translate the code to a modern language. This costs thousands of dollars and is prone to operator error. Rather than directly rewrite the COBOL code in one program, the purpose of this project is to create a reproducible framework that can identify the structure of COBOL code and map it onto a modern OOP design. Unlike existing tools which are purpose-built for one specific pipeline of code, this project aims to use AI automation to design a framework for system reference. This framework can be applied to any COBOL-based system and identify which procedures can be mapped into classes, creating a fully automated approach. This can work in tandem with existing structures and will be reproduced to any pipeline by exposing classes rather than rewriting them. To address challenges with existing tools, the framework is designed to use unsupervised learning to expose classes and provide targeted guidance of what possible solutions may be used. Additionally, we train the model on a large library of existing code for both Python and COBOL for maximum exposure in predicting classes. It is recognized that C++ is the gold standard for converting legacy code for enterprise systems due to its resource efficiency and flexibility. However, Python was chosen for this prototype because it has the most extensive collection of libraries for converting legacy code. In addition, output code can be of any language, and the framework creates only stubs as opposed to full programs, significantly reducing the strain on resources. Extensive experiments on key Github datasets demonstrate that the proposed method effectively identifies the desired classes and achieves complete functional equivalence to existing solutions. This framework also functions as a plug-and-play solution for mainstream legacy code conversion. Code is available at https://github.com/terrasky064/cs422-odu-spring26-cobol-oop-modernization.

Legacy systems introduce significant challenges, including security vulnerabilities, compatibility issues with modern cloud infrastructure, high maintenance costs, and

substantial technical debt. Modernizing these systems is difficult due to complex dependencies, accumulated business logic spanning decades, and a shrinking pool of specialized programmers. The societal problem is clear: organizations face increasing risks of system failures, security breaches, and inability to leverage modern technologies while being locked into expensive, hard-to-maintain codebases.

The needed solution characteristics include accurate code translation that preserves functional equivalence and business logic, dependency analysis, support for incremental modernization, integration with modern development practices, and mechanisms to reduce errors in AI-assisted conversion. Legacy Lift is the proposed AI-driven pipeline solution. It analyzes legacy source code (primarily COBOL), generates structured outputs describing components and relationships, and supports incremental conversion to modern languages like Python 3.11. By leveraging Retrieval-Augmented Generation (RAG) with Gemini AI, it maintains compatibility while enabling security, performance, and cloud-readiness benefits of modern platforms.

This document outlines the Legacy Lift product, its features, major components, target users, and supporting materials as part of the CS 410 prototyping effort.

2. Legacy Lift Product Description

Legacy Lift is an AI-assisted legacy code analysis and modernization toolchain. It ingests legacy source code, performs automated parsing, dependency mapping, and data flow analysis, then uses Retrieval Augmented Generation (RAG)-enhanced AI (Gemini) to generate functionally equivalent modern Python code. The tool aims to reduce the risks and technical debt of legacy systems modernization while preserving critical business logic.

Goals and Objectives:

- Provide functional equivalence between legacy and modern code.
- Reduce technical debt and maintenance costs.
- Enable safe migration to cloud-native environments.
- Support consultants and organizations modernizing enterprise systems.

2.1 Key Product Features and Capabilities

Legacy Lift performs the following:

- Automated Parsing & Tokenization of legacy code (COBOL focus).
- Dependency Mapping, Data Flow Analysis, and Architecture Summary.
- RAG Retrieval to provide context-specific legacy code, documentation, and patterns to the AI model, significantly reducing hallucinations and preserving business logic.
- AI Code Generation/Translation (Gemini) that produces clean, readable Python 3.11 code with modern structure (functions, OOP-ready).
- Change Documentation and Logging of modifications.
- System Testing for compatibility and functional equivalence.
- Dockerized Deployable Environment for easy demonstration and use.

Significance and Innovation:

Legacy Lift achieves substantial code reduction (e.g., 60+ lines of COBOL input validation reduced to ~15 lines of Python) while maintaining structure and readability. It uniquely combines RAG for accuracy with human-reviewable outputs, differing from fully automated tools by emphasizing verifiable equivalence and incremental modernization. It addresses both translation and infrastructure modernization. This solves the core problem by making modernization faster, safer, and more accessible.

Feature Comparison (Table 1):

FEATURES	Legacy Lift	Chat GPT	IBM WatsonX Code Assistant	Azure Migrate
Legacy Code Conversion	✓	✓	✓	✓
Dependency Analysis	✓			✓
Full Compatibility with Plugins or Frameworks	✓			
AI Incremental Translation Pipeline	✓			
New Code has Functional Equivalence with old code	✓		✓	✓
Audit log of changes	✓			✓
Modern OS/x86 Compatibility	✓		✓	✓

2.2 Major Components (Hardware/Software)

Hardware/Deployment:

The solution runs on standard modern x86 systems (Windows 11 or Ubuntu 24.04). It is packaged as a self-contained Docker environment for portability and cloud deployment. No specialized legacy hardware is required post-migration.

Software Architecture (based on CS 410 Major Functional Components Diagram - MFCD):

- Frontend: Python TKinter widgets with Electron wrapper for cross-platform GUI.
- Backend: Python 3.11 core with Gemini AI integration.
- Storage: MySQL Server + Vector Database (for RAG retrieval of code chunks, documentation, and patterns).
- Development Tools: Visual Studio Code, Git/GitHub, Miniconda, Docker.
- Key Processes: Parsing/Tokenization, Dependency Mapping, RAG Retrieval, AI Processing, Testing/Validation.

Major Functional Components:

- Code Ingestion & Parsing
- Analysis Engine (Dependencies, Data Flow)
- RAG Layer
- AI Translation Engine
- Testing & Validation Suite
- Logging & Documentation Generator

The system follows an Agile development model with clear deliverables including updated Python code, change logs, test suites, and the Docker environment.

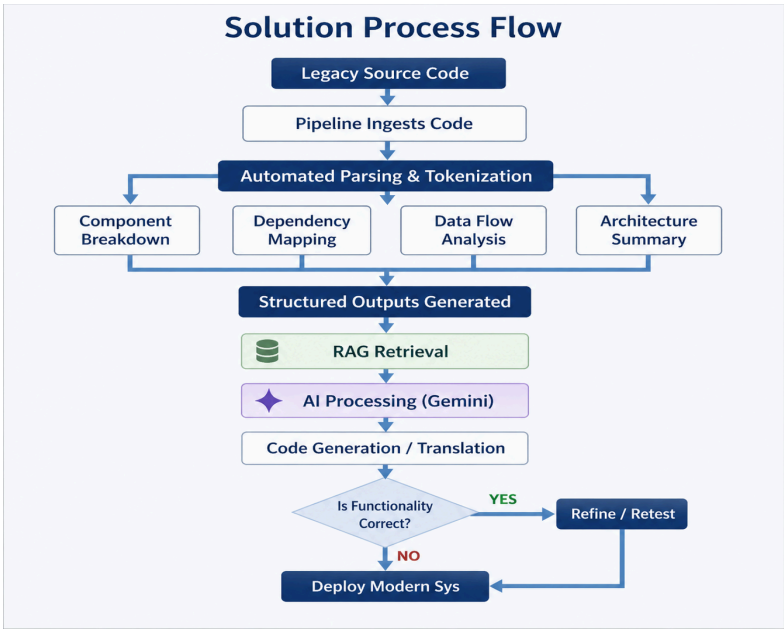

Figure 1: Solution Process Flow (Detailed diagram showing pipeline from Legacy Source Code through AI Processing to Deploy Modern Sys).

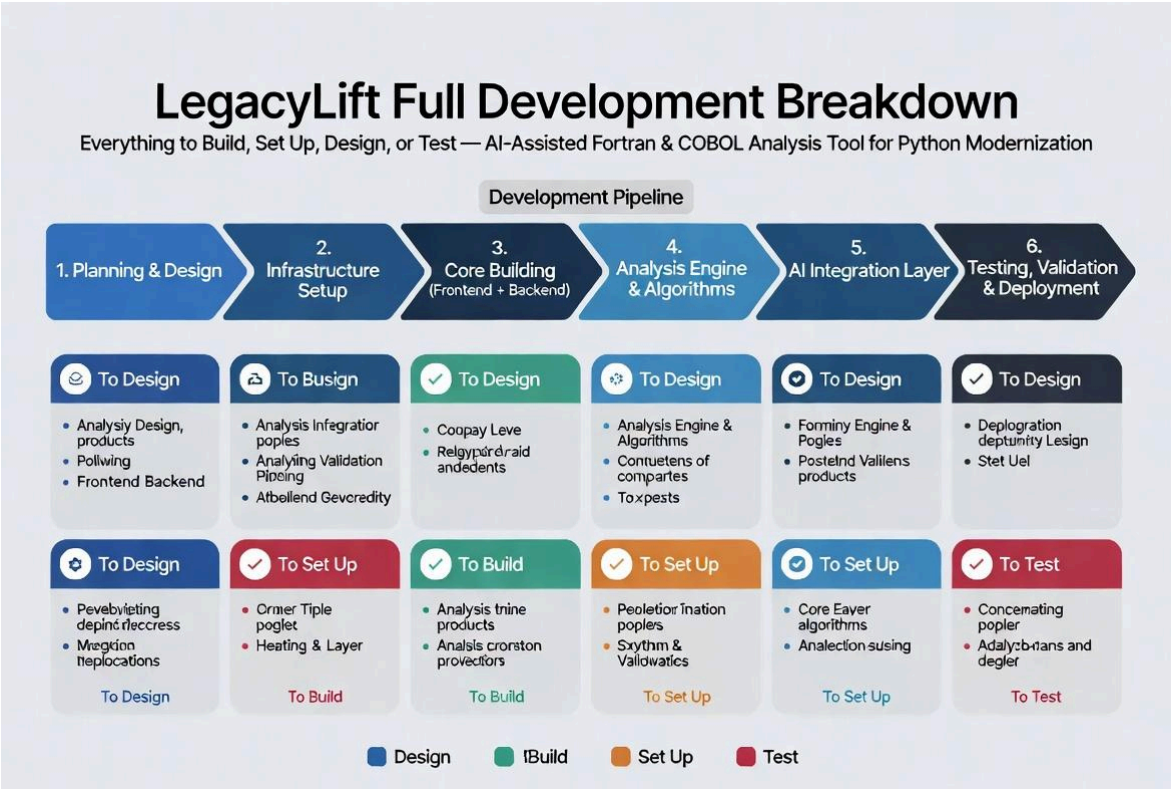

Figure 2: Work Breakdown Structure (Development Pipeline phases).

3. Identification of Case Study

Legacy Lift is primarily developed for consultants and IT professionals in organizations with legacy COBOL/Fortran systems (banks, government agencies, hospitals, insurance). These users need a cost-effective, low-risk way to modernize without full rewrites or losing business logic.

The primary reason being that specialist legacy programmers are scarce and expensive ($150+/hr), while modern Python developers are abundant. Legacy Lift bridges this gap, lowering maintenance costs and security risks.

Future Users:

- Enterprise development teams migrating to cloud.
- Small-to-medium businesses seeking modernization.
- Insurance providers and banking firms with large legacy datasets
- State and Local government agencies
- Open-source contributors extending support for additional languages.

4. Glossary

COBOL – Common Business-Oriented Language; a language that was popular for corporate mainframes in the 1970s and 80s but has largely become obsolete with OOP

Docker – Platform for containerizing applications.

Functional Equivalence – New code produces identical outputs/behavior to legacy code for the same inputs.

Gemini – Google's AI model used for code translation.

Hallucination - An error within an LLM where it produces output that is logically incorrect but valid to an LLM due to limited data; akin to an educated but incorrect guess

IDE - Integrated Development Environment; collection of tools that allow unified coding, debugging and production within a single application

LLM - Large Language Model; a software framework capable of independent reasoning based on massive public datasets; the core component of Legacy Lift

OOP - Object Oriented Programming; development style that organizes code via modular segments called classes as opposed to linear blocks of code called procedures

Python - an interpreted language commonly used in machine learning

RAG – Retrieval-Augmented Generation; technique combining information retrieval with generative AI for improved accuracy.

SQL - Structured Query Language; the default format for enterprise databases like MySQL and PostGresSQL.

Technical Debt – Accumulated cost of maintaining outdated code; Inherent in legacy systems, especially ones that require specific hacks or knowledge of a specialized pipeline to operate

Vector Database – Database optimized for semantic search of embeddings (used in RAG); is usually faster and more accurate than randomly uploading files to an LLM; trained on the content specific to the product's field of production

5. References

- “12 Best Legacy Modernization Tools for Companies in 2026.” Launchpad, 2 Mar. 2026, launchpad.io/blog/best-legacy-modernization-tools.
- “2025 Legacy Code Stats.” Pragmatic Coders, 2 Sept. 2025, www.pragmaticcoders.com/resources/legacy-code-stats.
- *The 2025 State of Legacy Software Modernization Report*. Saritasa. Accessed 11 Mar. 2026.
- “44 Legacy System Modernization Statistics Every Enterprise Should Know in 2026.” DreamFactory. Accessed 11 Mar. 2026.
- Ellienosrat, et al. “Best Practices for Leveraging Azure OpenAI in Code Conversion Scenarios.” Microsoft Tech Community, 26 Mar. 2025.
- “IBM Watsonx Code Assistant for Z.” IBM, 12 Apr. 2024, www.ibm.com/products/watsonx-code-assistant-z.
- “Lost in Translation: What the AI Code Debate Keeps Getting Wrong.” IBM Newsroom. Accessed 11 Mar. 2026.
- Sabrina. “Legacy Software Modernization in 2025: Survey of 500+ U.S. IT Pros.” Saritasa, 25 Aug. 2025.
- “The Dark Side of AI: Assessing Liability When Bots Behave Badly.” EBG Law, 22 Sept. 2025.

Images

FEATURES	Our Product	ChatGPT	IBM Watsonx Code Assistant	Azure Migrate
Legacy Code Conversion	✓	✓	✓	✓
Dependency Analysis	✓			✓
Full Compatibility with Plugins or Frameworks	✓			
AI Incremental Translation Pipeline	✓			
New Code has Functional Equivalence with old code	✓		✓	✓
Audit log of changes	✓			✓
Modern OS/x86 Compatibility	✓		✓	✓