
Legacy Lift
Legacy Lift
LEGACY LIFT
aws
Legacy Lift
AI
Yesterday's Problem → Tomorrow's Solution

Table of Contents

Meet the Team

Spencer Peloquin - He is enrolled in the BSCS MS program at Old Dominion University. He has completed a technical internship at Tacoma Public Utilities. In addition, he has experience in Python, Java, Ruby, C++, C#, SQL and Fortran. He lives in Tacoma WA, where he enjoys running, reading and hiking.

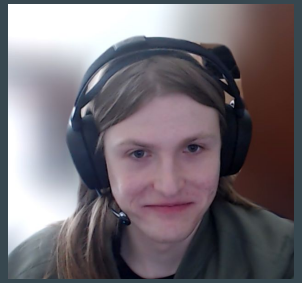
Aaditya Ghimire(AG)- He is enrolled in Computer Science Program at ODU. He has experience in Java + Spring Boot and with main focus on Python deep learning computer vision.

Darius Itam - He is enrolled in the bachelors computer science program as a senior at ODU main campus in Norfolk, VA. He is proficient in Python, Java, JavaScript, and C#. He loves fitness, esports, and music.

Caleb Peterson - He is enrolled in the bachelors computer science program at ODU. He has experience in Java, JavaScript, C++, and SQL. His hobbies include video games and board games.

Justin Tendilla - CS Major. Loves playing guitar, video games, and movies. Experienced in C++ and Java.

Jordan Kemelhar - IT/Azure cloud professional and Senior CS student at Old Dominion University. Certified in CyberSecurity Has experience in Java and Azure user management. Hobbies video games, and, hiking, board games.



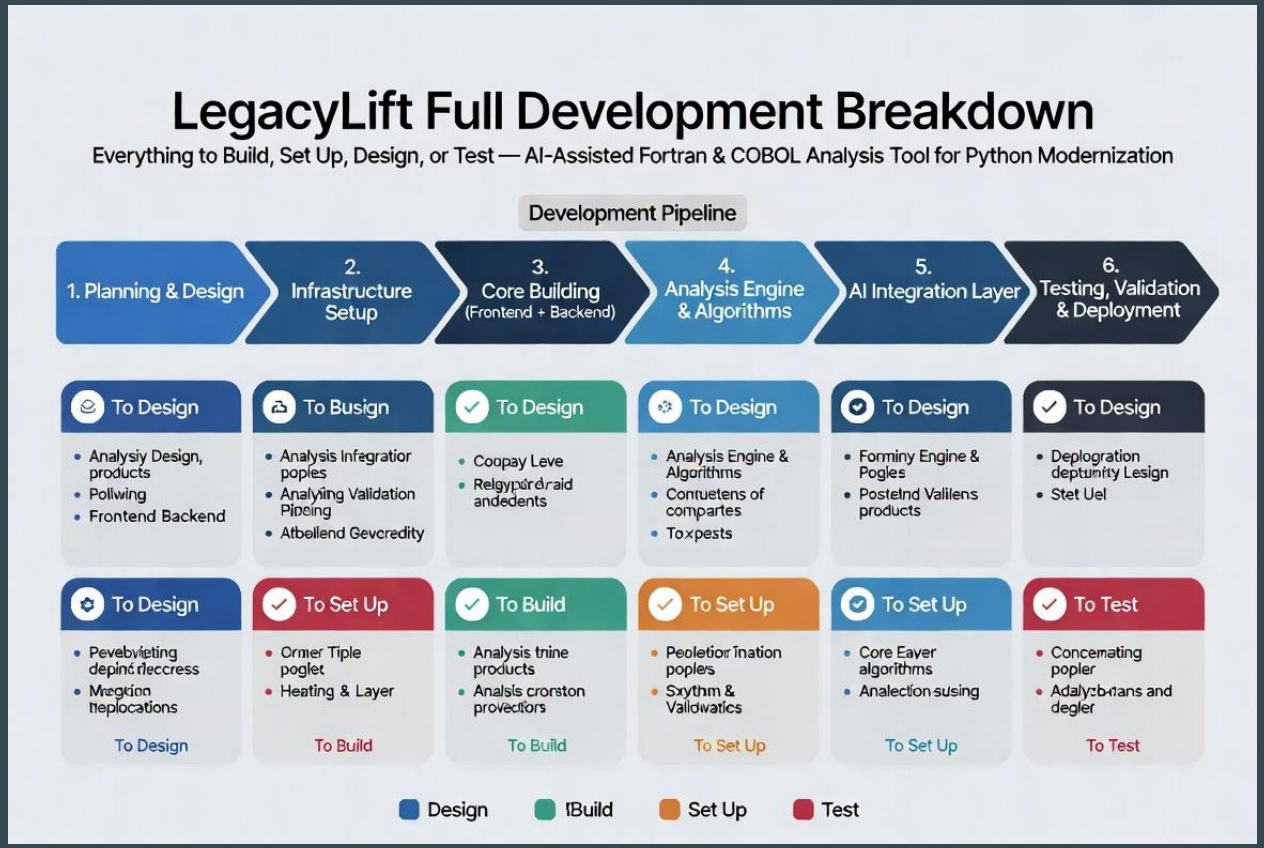
Executive Summary
AI Pipeline for Legacy Code Modernization

Problem
Many critical enterprise systems still run on legacy software written in outdated languages such as COBOL. These systems are deeply embedded in industries such as finance, government, and enterprise infrastructure. 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).

Impact
The problem is that legacy systems introduce security risks, compatibility issues with modern technology, and large amounts of technical debt. Modernizing them is extremely difficult because these systems contain complex dependencies and decades of accumulated business logic. It is especially important to port code to modern platforms for use in cloud applications, which are primarily incompatible with legacy systems.

Solution
Legacy Lift proposes an AI-driven pipeline that analyzes dependencies and incrementally converts legacy systems to modern infrastructure. It will maintain full compatibility with the infrastructure of the legacy code while enabling the security/performance benefits of modern programming languages.

Work Breakdown Structure



High Level Overview

Purpose:

- Translate legacy code from COBOL to modern Python 3.11
- Solve technical debt with legacy codebases
- Resolve ongoing security and memory issues with legacy systems

Mechanics:

- Uses Gemini AI to convert code
- Translates legacy plugin dependencies into modern package hooks
- Preserves class structure from original code

Benefits:

- Functional equivalent code conversion
- Leverages both Generative AI and human review
- Provides an on-ramp to modern systems

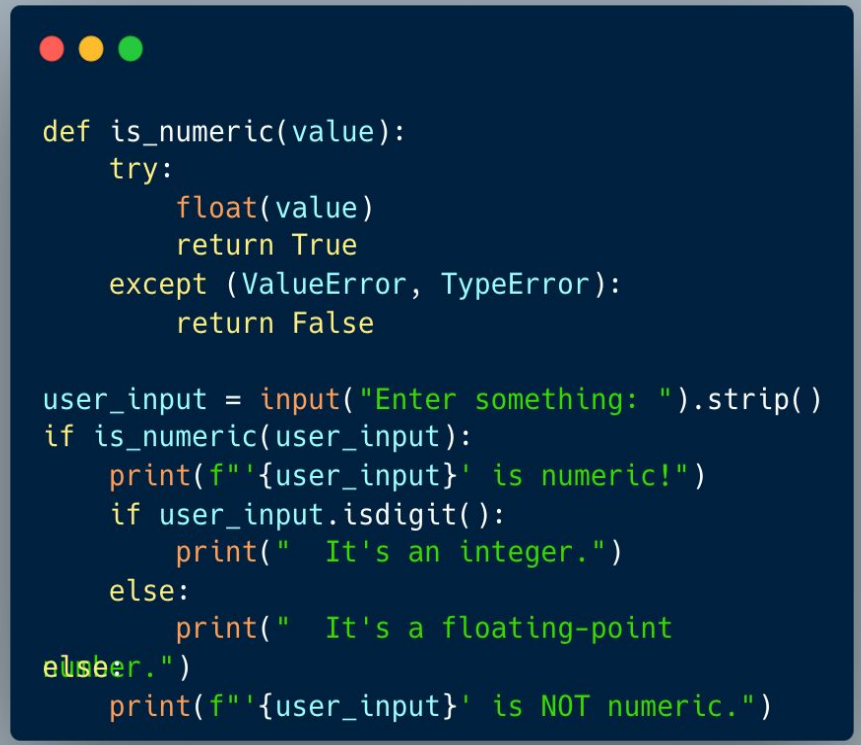
Code Comparison- Cobol

- 60+ lines for a single input validation task
- No OOP — paragraph-based procedural flow
- Fixed-width memory declarations (PIC X(10)

```
identification division.
program-id. is-numeric-test.
data division.
working-storage section.
01 ws-user-input pic x(10).
01 ws-user-input-justified pic x(10) justified right.
procedure division.
main-procedure.
    perform process-plain
    perform process-zero-fill
    perform process-trim
    stop run.
process-plain.
    display "(plain) Enter a value: " with no advancing
    accept ws-user-input
    if ws-user-input is numeric then
        display ws-user-input " is numeric!"
    else
        display ws-user-input " is not numeric."
    end-if
    exit paragraph.
process-zero-fill.
    display "(right justify, zero fill) Enter another value: "
        with no advancing
    accept ws-user-input-justified
    inspect ws-user-input-justified
        replacing leading spaces by '0'
    if ws-user-input-justified is numeric then
        display ws-user-input-justified " is numeric!"
    else
        display ws-user-input-justified " is not numeric."
    end-if
    exit paragraph.
process-trim.
    display "(trim) Enter a third value: " with no advancing
    accept ws-user-input
    if function trim(ws-user-input) is numeric then
        display function trim(ws-user-input) " is numeric!"
    else
        display function trim(ws-user-input) " is not
numeend-if
    exit paragraph.
end program is-numeric-test.
  return go(f, seed, [])
}
```

Code Comparison - Python

- 15 lines — 75% reduction
- Clean function-based OOP-ready structure
- Readable without COBOL-specific knowledge



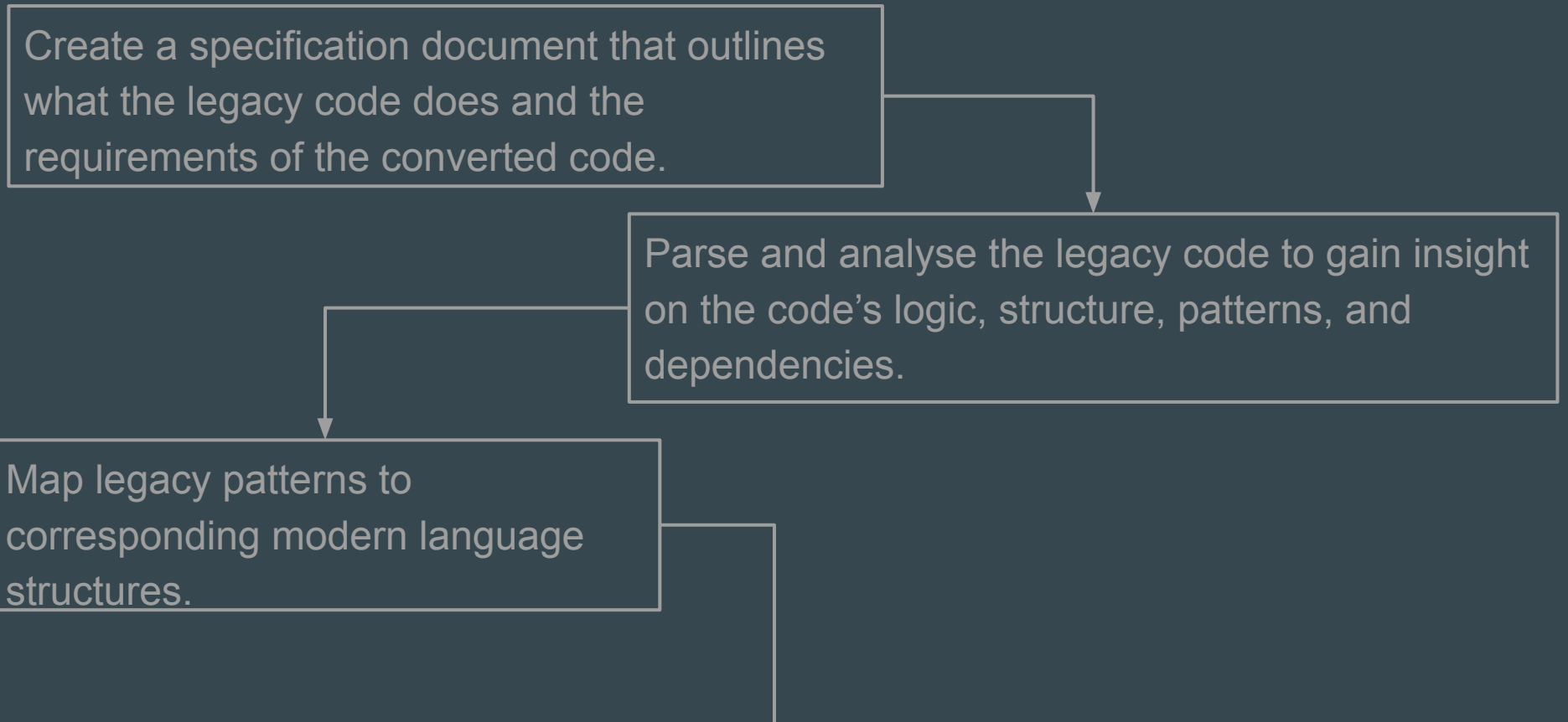
```python
def is_numeric(value):
    try:
        float(value)
        return True
    except (ValueError, TypeError):
        return False

user_input = input("Enter something: ").strip()
if is_numeric(user_input):
    print(f"'{user_input}' is numeric!")
    if user_input.isdigit():
        print("  It's an integer.")
    else:
        print("  It's a floating-point
else:
    print(f"'{user_input}' is NOT numeric.")
```

Infrastructure Comparison

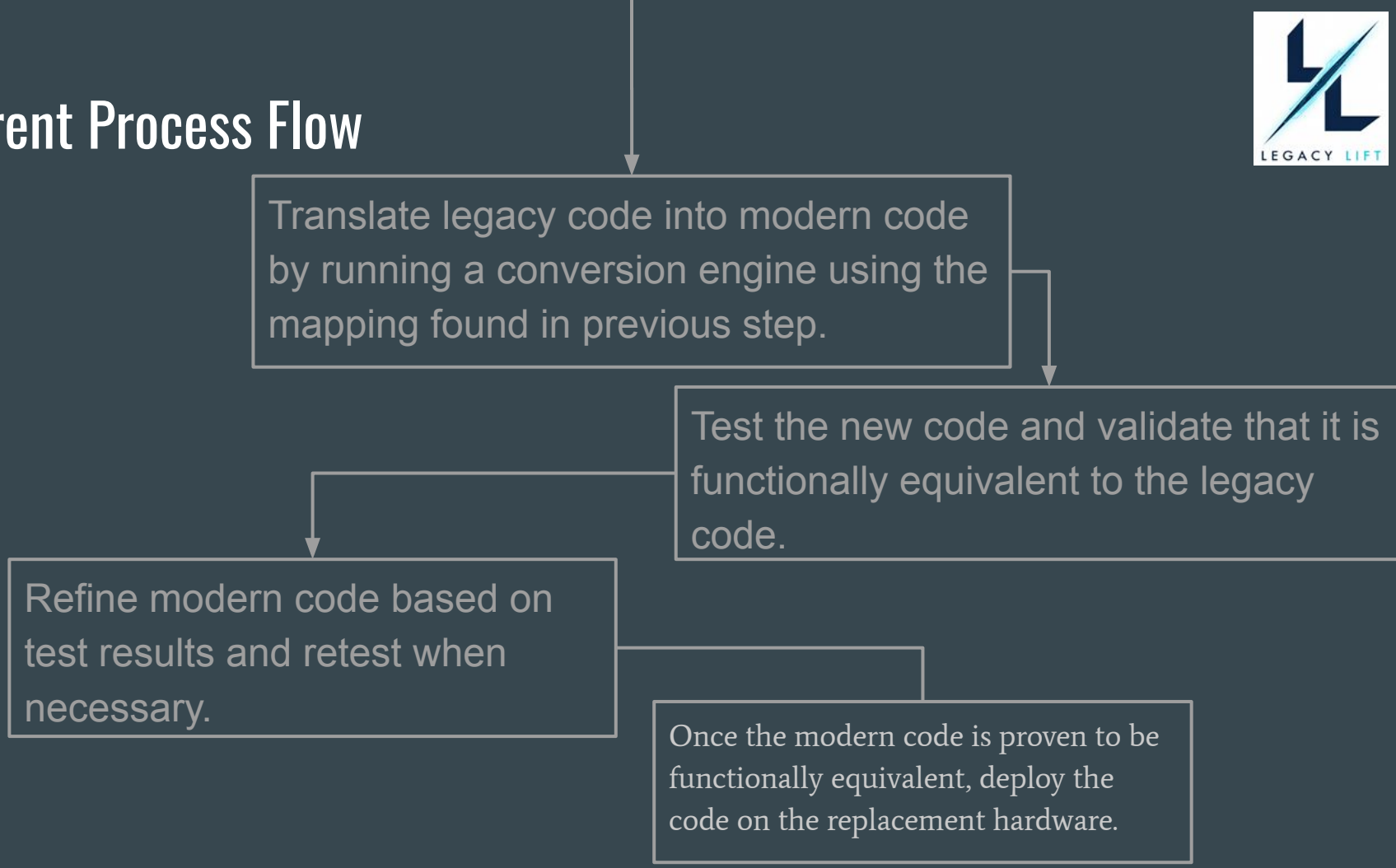
Feature	Legacy Code (COBOL/Ada)	Modern Python Environment
Language	COBOL, Fortran, Ada	Python 3.11+
Developers	**Scarce**	**Millions:** Most taught globally
IDE / Tooling	Proprietary, limited debuggers	VSCode, linters, extensions
Testing	Manual, minimal frameworks	PyTest, Automated CI pipelines
Documentation	Often missing or non-standard	Docstrings, Auto-generated docs
Version Control	None or proprietary systems	Git + GitHub + GitHub Actions
Security	Slow patches, legacy vulnerabilities	Active community, rapid patching
Maintenance Cost	**High:** Specialist rates ($150+/hr)	**Standard:** Competitive market

Feature	RWP	Legacy Lift
COBOL → Python 3.11 Translation	✔	✔
Fortran → Python Translation	✔	—
Ada → Python Translation	✔	—
Recreate Original Class Structure	✔	—
Modern Design Refactoring	✔	✔
Runs on Modern x86 Systems	✔	✔
Plugin / Framework Compatibility	✔	—
Preserve Comments & Documentation	✔	✔
Audit Logging	✔	—
Functional Equivalence	✔	✔
Dependency Analysis	✔	✔
Multi-file / Large Codebase Support	✔	—
CI/CD Integration (Enterprise Scale)	✔	✔
Security Vulnerability Scanning	✔	—

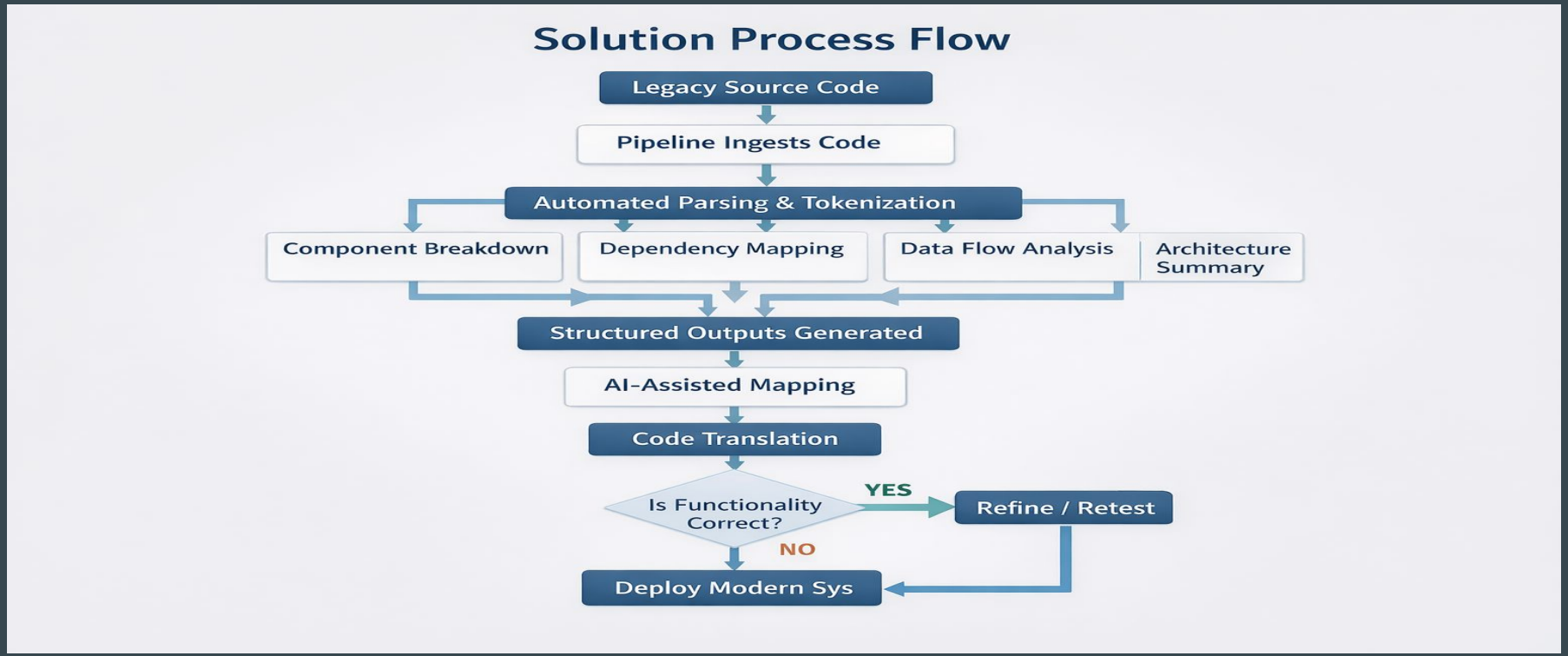
Current Process Flow



Current Process Flow



Solution Process Flow



Solution Statement

This project will produce a prototype software pipeline that analyzes legacy source code and generates structured outputs describing the code's components and relationships, demonstrating the feasibility of using AI-assisted techniques to support legacy system modernization.

Software Type

- AI-assisted legacy code analysis tool

Purpose

- AI-assisted analysis of legacy source code
- Marketed for Consultants.

High-level Function

- Accept legacy code as input
- Analyze structure and dependencies
- Produce structured outputs describing the system

Features

- Translate legacy code from COBOL (initially) to Python
- Recreate the class structure of the original code
- Refactor code to match modern design protocols
- Enable translated code to run on a modern x86 PC Operating System
- Contain full compatibility with any existing plugins or frameworks
- Translated code will be fully equivalent in function to the original code
- All comments and documentation will be intact
- Contain an audit log of all changes made to the original code

What it Will Not Do

- Be able to run unmodified on legacy hardware
- Run its own code
- Translate or decompile binary modules
- Retain any or all technical debt
- Assume missing functionality
- Access the network or external/cloud drives
- Guarantee the absence of bugs or security issues
- Convert legacy code into code of a completely different nature
- Function as a chatbot, game engine or “swiss army knife tool”

Competition Matrix

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	✓		✓	✓

Competition Analysis

Current competition includes IBM and Microsoft. These models use in house AI conversion tools based on Copilot models. These tools also come with enterprise support and are open source. The key difference between our model (Legacy Lift) and theirs is that our model uses the Gemini model. It is fully backed by a University CS department and does not collect data for sale. Other research projects from universities like Stanford and MIT hold this model, and should provide proper benefits for a company.

Major Functional Components

Tech stack:

L - Container - Docker

A - Cloud - Google GCP

M - DB - MySQL

P - Language - Python

Python CLI -> Provides the Conda shell to run the code

Program runs -> The Legacy Lift Software is initialized on the cloud PC

Code Parsing -> The Legacy Lift Software takes tokens and classes and sends them to Gemini

Sends to Gemini -> Gemini parses these tokens and generates compatible classes

Dependency analysis -> The tokens are analyzed for possible dependencies and compatible classes

Converts Code to target language -> These classes are rebuilt for Python with a focus of creating modern code over preserving legacy quirks

Verify integrity -> check for corruption and verify all classes are valid

Makes sure it can run -> run the first 10,000 tokens on the bare metal for stress testing

Validates one to one functionality -> A series of unit, system and integration tests are run to ensure 1:1 compatibility

Deploys new code -> create a Conda Environment to deploy new code

Algorithms

Parsing & Abstract Syntax Tree (AST) Construction

Component Detection & Inventory

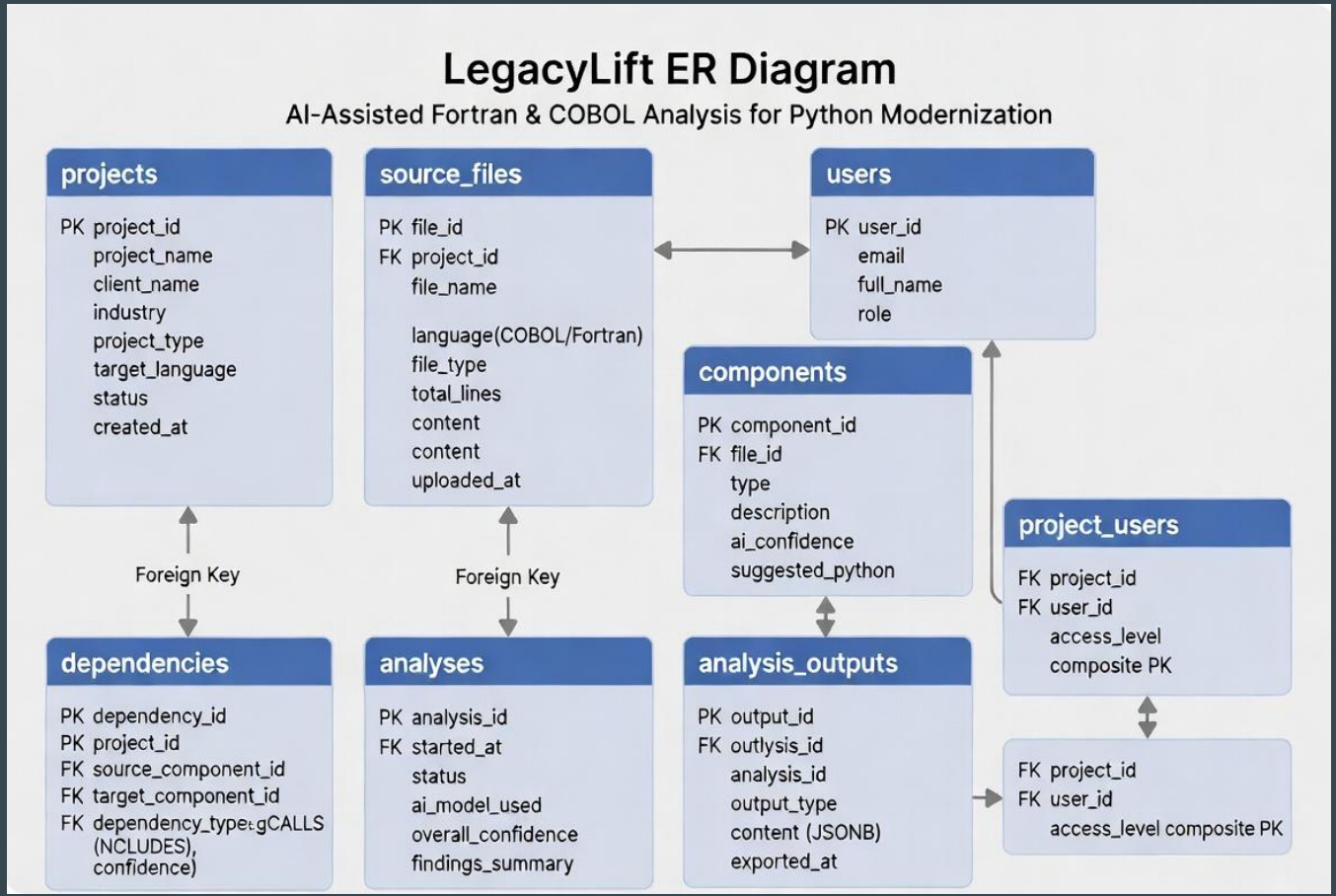
Dependency & Relationship Analysis

AI-Assisted Analysis Layer

Graph & Visualization Algorithms

Supporting / Utility Algorithms

Database Schema



Legal Risks

- **Primary Risk** : Users over-trust AI in the COBOL-to-Python translator, assuming consistent accuracy despite frequent hallucinations in code logic, data handling, or refactoring.
- **Real-World Evidence** : Numerous lawsuits and sanctions against AI users (e.g., lawyers fined $1,500–$86,000 for hallucinated legal citations); enterprise hallucination losses estimated at **$67.4 billion** in 2024.
- **Probability** : High — AI confidently wrong outputs are common (e.g., 17–34% hallucination rates even in specialized legal AI tools).
- **Impact Rating** : 4/5 — Heavy financial (fines, rework, lost revenue) and reputational costs for a small modernization startup.
- **Specific Relevance** : Critical for legacy migrations where errors in translated COBOL business rules or Fortran numerical code can cause production failures or compliance issues.
- **Key Recommendation** : Strong disclaimers, mandatory human review, and clear limitation disclosures in all client engagements to reduce liability.

Risk Analysis

- Incompatibility & Technical Debt
- Compatibility between legacy and modern Python
- Sustainable Revenue Stream/Cash Flow
- Migration Failure Risks
- AI Over-Trust Amplification
- Implementation and Support Challenges

Project Monetization

Discovery and Assessment

Initial evaluation costs range from $2,000 to $10,000, providing tailored project insights for clients.

Framework Implementation

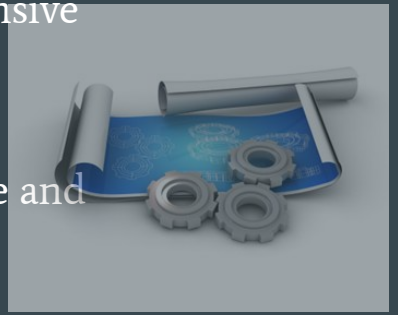
Implementation services are priced $5,000 to $20,000, reflecting increased complexity and customization.

Full Migration

Comprehensive migration services cost $20,000 to $80,000, covering extensive technical work and support.

Annual Maintenance

Ongoing maintenance is $2,000 per year, ensuring continued performance and reliability for clients.



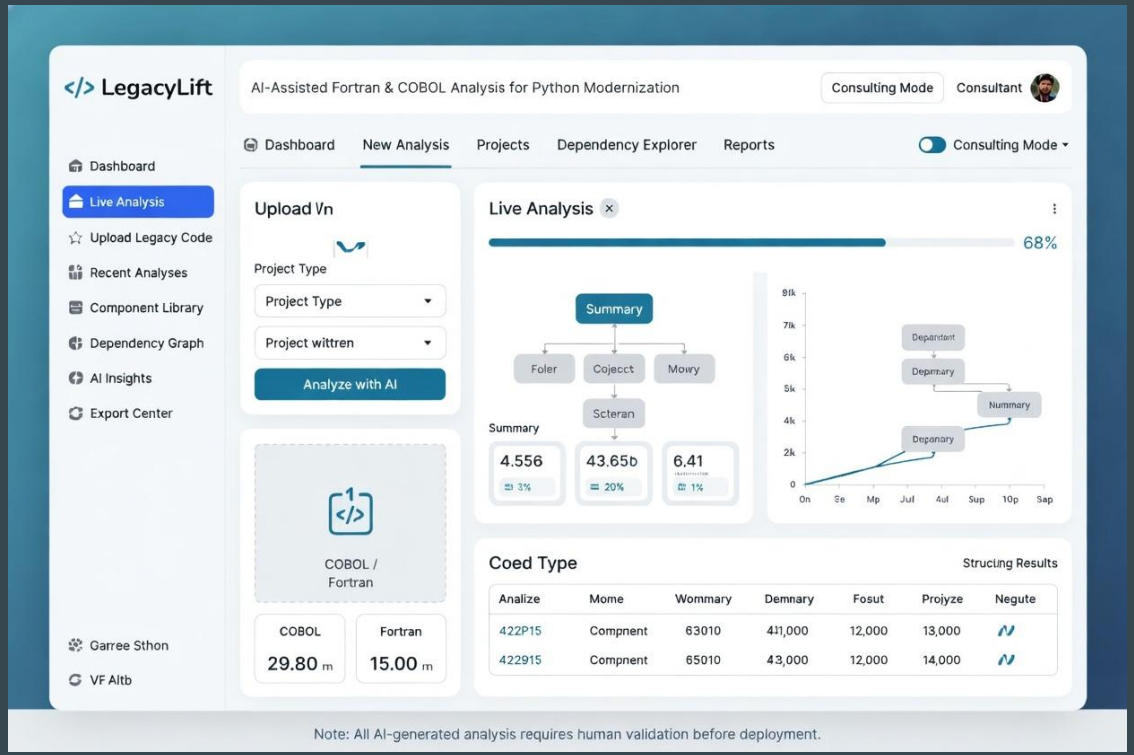
Development Tools

- Visual Studio Code
- Python 3.11
- Windows 11/Ubuntu 24.04
- Miniconda
- MySQL Server
- GitHub + Git
- Gemini
- Docker

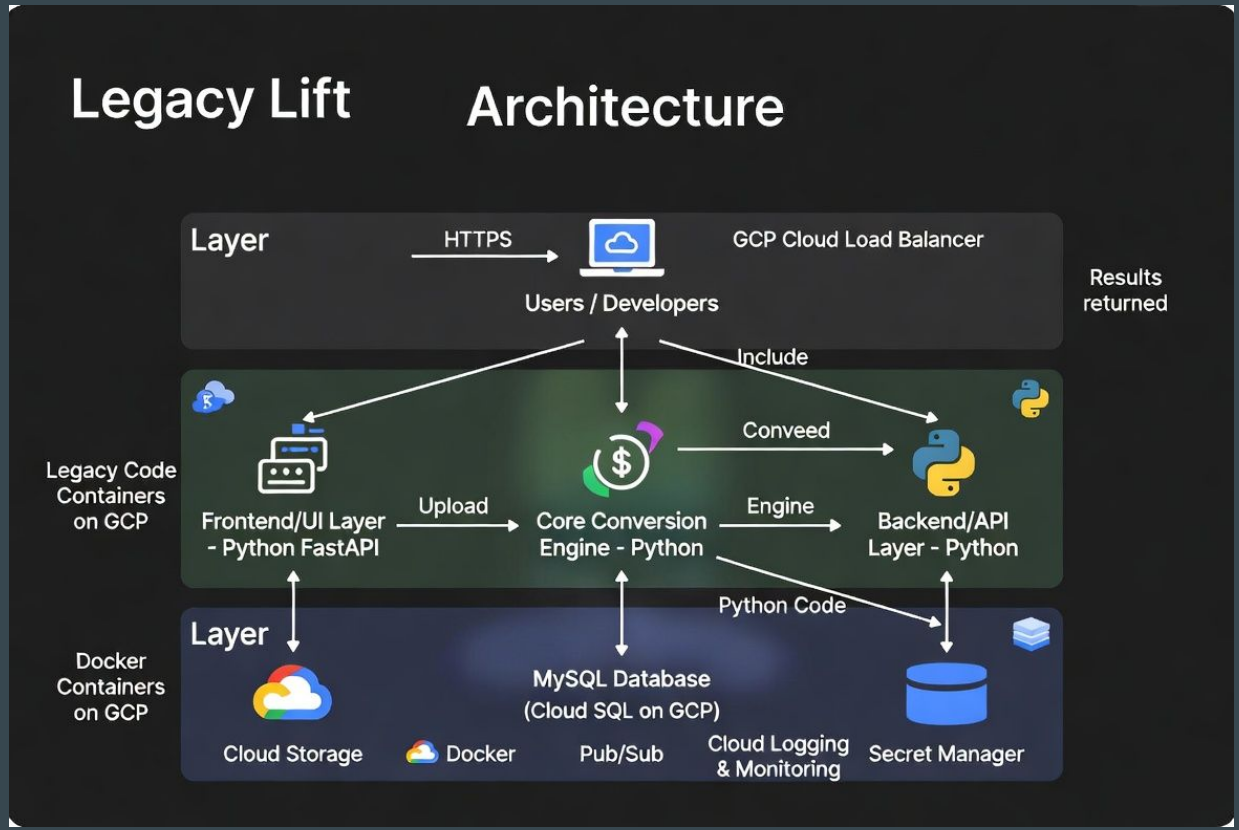
Libraries and Frameworks

- FastAPI - API Framework
- Psycopg2 - PostGreSQL driver
- ANTLR4 Python runtime - COBOL/Fortran Parsing
- NetworkX - Network Graphing
- LangChain - Gemini Integration
- Python-dotenv - Environment Management
- React - Frontend
- Pytest - Unit testing
- f2py / numpy / scipy - Data Manipulation

Project Mockup



Major Functional Components Diagram



Major Functional Components Diagram

Conclusion

Legacy Lift is a viable solution for small businesses seeking a stable, cost effective means for updating and modernizing their legacy code infrastructure.

Questions?

References

"12 Best Legacy Modernization Tools for Companies in 2026." *12 Best Legacy Modernization Tools for Companies in 2026*, 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, www.saritasa.com/wp-content/uploads/2025/07/Saritasa-2025_State_of_Legacy_Software_Modernization_Report.pdf. Accessed 11 Mar. 2026.

"44 Legacy System Modernization Statistics Every Enterprise Should Know in 2026." *DreamFactory [LIVE]*, www.dreamfactory.com/hub/legacy-system-modernization-statistics. Accessed 11 Mar. 2026.

Ellienosrat, et al. "Best Practices for Leveraging Azure Openai in Code Conversion Scenarios: Microsoft Community Hub." *TECHCOMMUNITY.MICROSOFT.COM*, 26 Mar. 2025, techcommunity.microsoft.com/blog/azure-ai-foundry-blog/best-practices-for-leveraging-azure-openai-in-code-conversion-scenarios/4395993.

"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*, newsroom.ibm.com/blog-lost-in-translation-what-the-ai-code-debate-keeps-getting-wrong. Accessed 11 Mar. 2026.

Sabrina. "Legacy Software Modernization in 2025: Survey of 500+ U.S.. It Pros." *Saritasa*, 25 Aug. 2025, www.saritasa.com/insights/legacy-software-modernization-in-2025-survey-of-500-u-s-it-pros.

"The Dark Side of AI: Assessing Liability When Bots Behave Badly." @Ebglaw, 22 Sept. 2025, www.ebglaw.com/insights/publications/the-dark-side-of-ai-assessing-liability-when-bots-behave-badly.

Appendix divider slide -