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

Process

~~All Feasibility Elements - Justin~~

~~Development Tools - Justin~~

~~Development Model (e.g., Agile) - Spencer - done~~

~~Identification/Description of deliverables - Jordan Kemelhar~~

Identification of Technical Risks and Customer Risks

- ~~Impact and probability - Spencer,Aaditya~~
- ~~Mitigations - Spencer,Aaditya~~

~~Major Functional Components - Jordan Kemelhar~~

- ~~Including COTS & Legacy Systems - Jordan Kemelhar~~

Technical Approach for software and hardware, including:

- ~~Dataflow diagrams - Spencer~~
- Process flow diagrams - Caleb
- Database schemas - Caleb
- ~~Data representation and management - Caleb~~

~~GUI Mockups and/or Rapid Prototypes - Spencer~~
~~Appendix - User Stories~~

Table of Contents

Table of Contents

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Many critical enterprise systems still run on legacy software written in outdated languages such as COBOL.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.

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

Development Tools

- Frontend
  - Python TK based widget
  - Electron wrapper
- Backend
  - Python 3.11
  - Gemini
- Database / Storage
  - MySQL Server
  - Vector Database for RAG retrieval
- Development / IDE
  - Visual Studio Code
- Version Control
  - GitHub + Git
- Environment / Infrastructure
  - Windows 11/Ubuntu 24.04
  - Miniconda
  - Docker

Deliverables

Updated Python code with functional equivalence to the legacy code.

Changes documentation with a collection of log files and original COBOL functions. RAG justification for python changes.

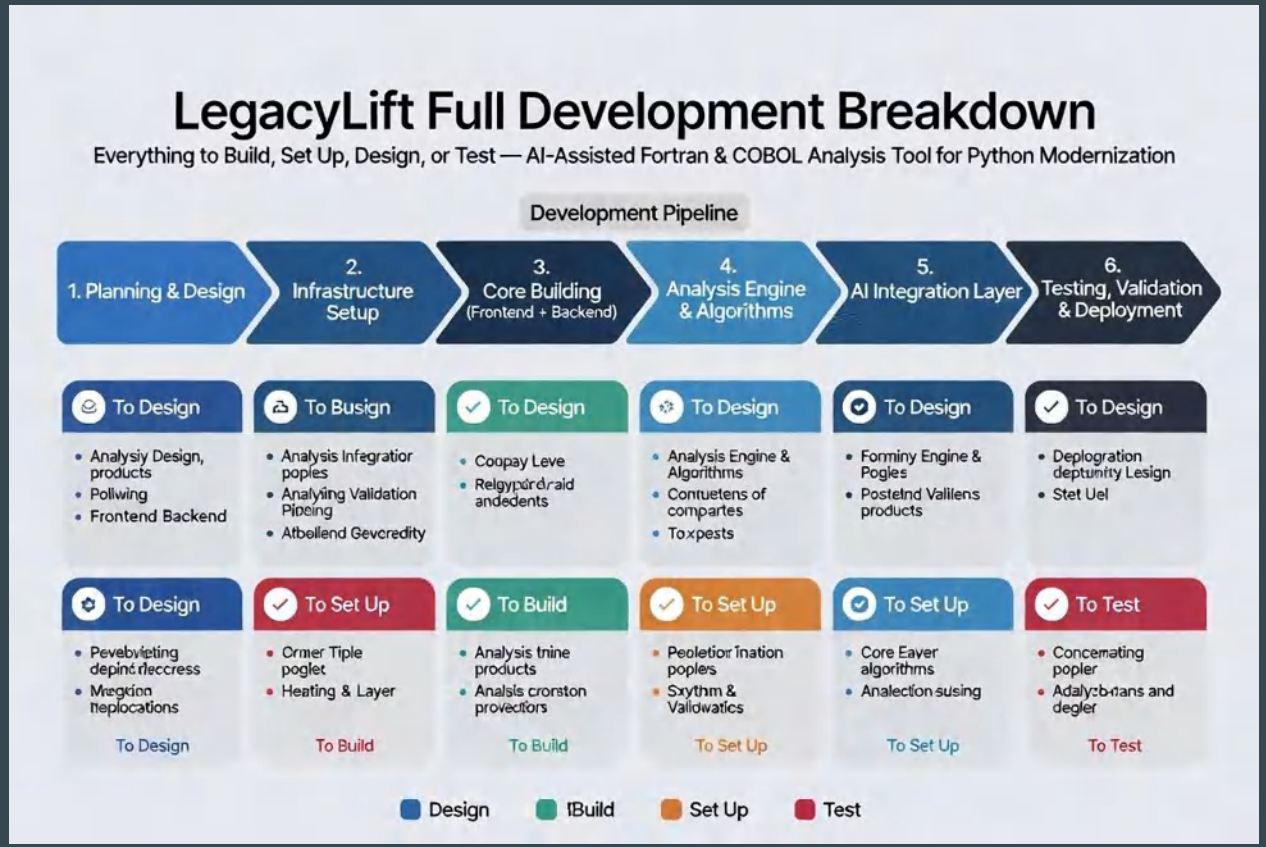
System tests to test for compatibility

Vector Database for Retrieval Augmented Generation

LegacyLift code update toolchain

Deployable self contained LegacyLift Docker environment for demo

Work Breakdown Structure



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 number.")
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	✔	—

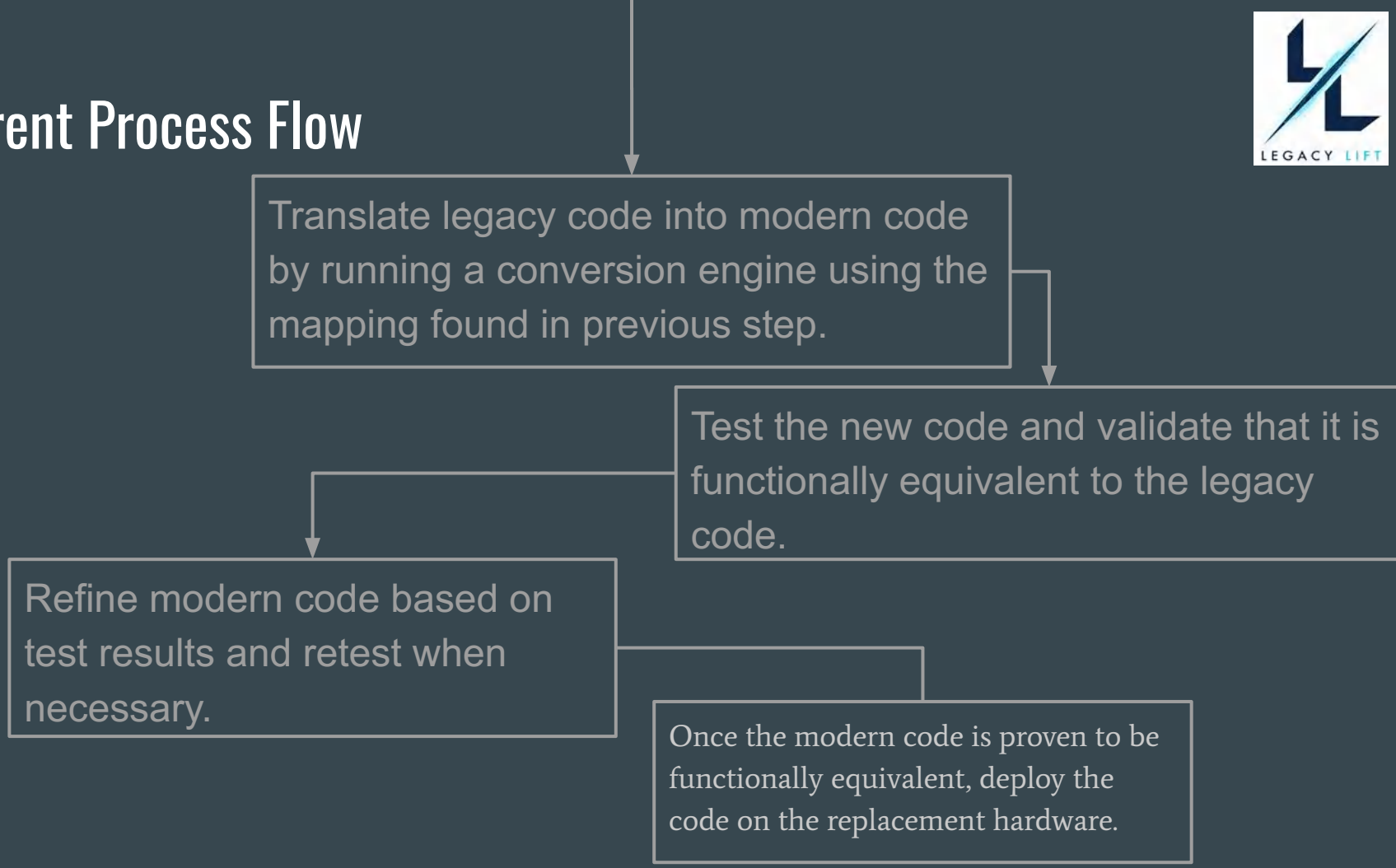
Current Process Flow

Create a specification document that outlines what the legacy code does and the requirements of the converted code.

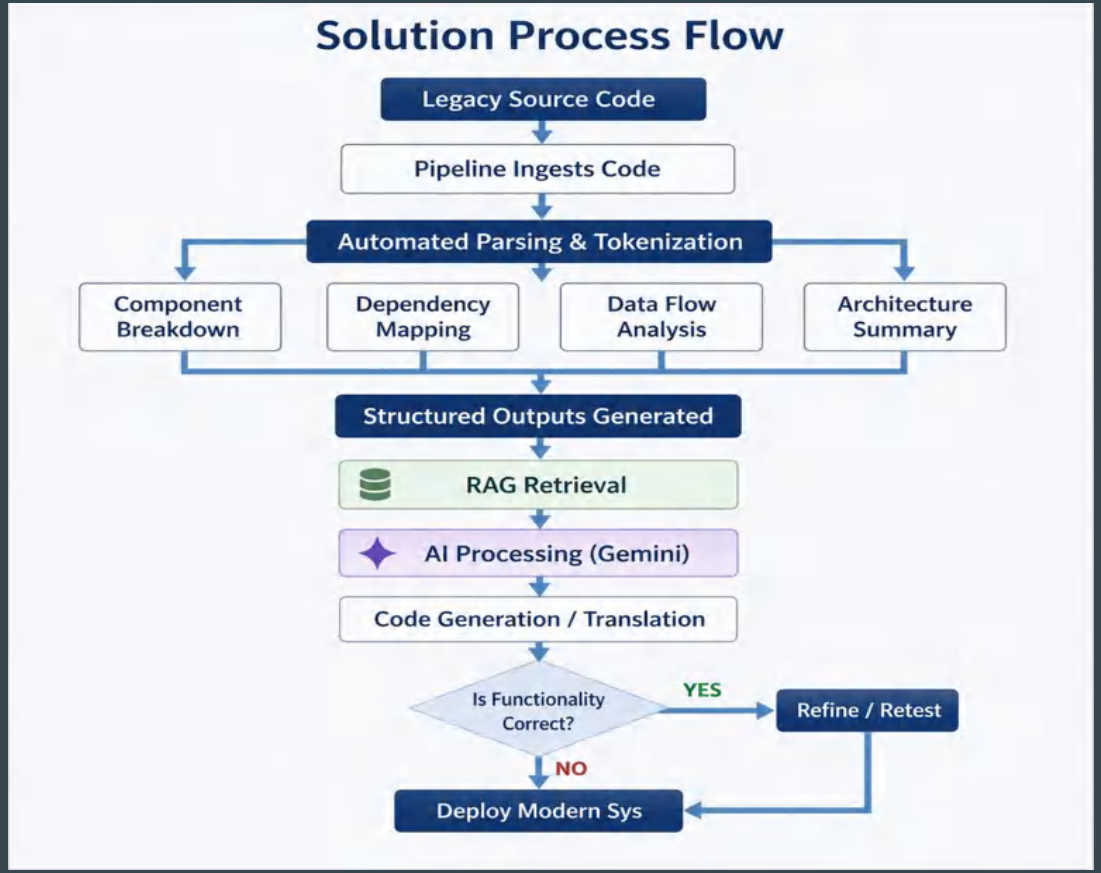
Parse and analyse the legacy code to gain insight on the code's logic, structure, patterns, and dependencies.

Map legacy patterns to corresponding modern language structures.

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
- Preserve program structure and translate it into modern Python modules
- 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

Limitations

- 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.
- They 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.

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
- Gemini produces hallucinated Python logic
- Migration Failure Risks
- AI Over-Trust Amplification
- Deployed code behaves differently in production

Mitigations:

- Disclaimers
- human review
- Audit trail of model version
- Runtime compatibility
- Full audit logging and documentation

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

COTS and Legacy Systems

COTS:

LegacyLift will use Gemini as the primary translation backbone. Retrieval Augmented Generation will be accomplished by using Gemini in a vector database.

MySQL will be the primary database implementation.

Uses GCP with docker so LegacyLift can be run anywhere with any code size

Legacy Systems:
Uses GnuCOBOL as an interpreter for COBOL

Agile Development Model

Agile development method

Chosen because LegacyLift works like a startup

Organized using a scrum

Waterfall and RUP are not suitable because LegacyLift is not enterprise

Algorithms

Parsing & Abstract Syntax Tree (AST) Construction

Component Detection & Inventory

Dependency & Relationship Analysis

AI-Assisted Analysis Layer

Graph & Visualization Algorithms

Supporting / Utility Algorithms

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

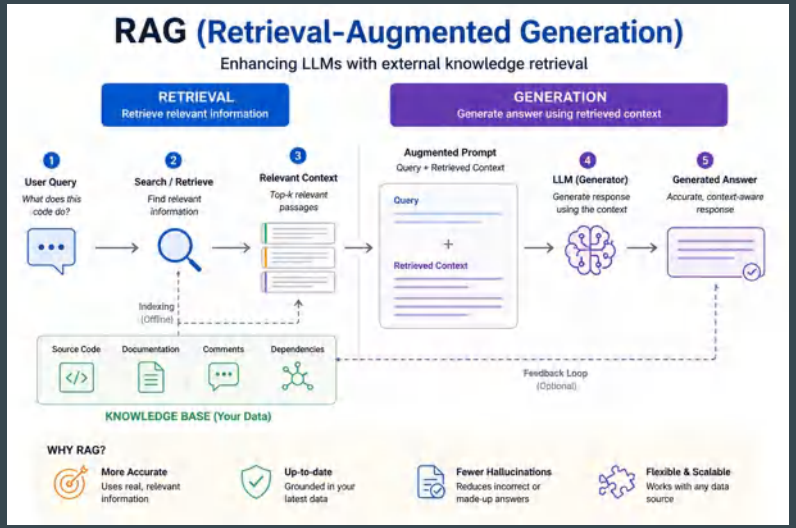
RAG - What It Is

Retrieval-Augmented Generation

- Combines data retrieval + AI generation
- Retrieves relevant code, documentation, and dependencies before AI processing
- Provides context to the model instead of relying on memory alone

RAG - How It Works

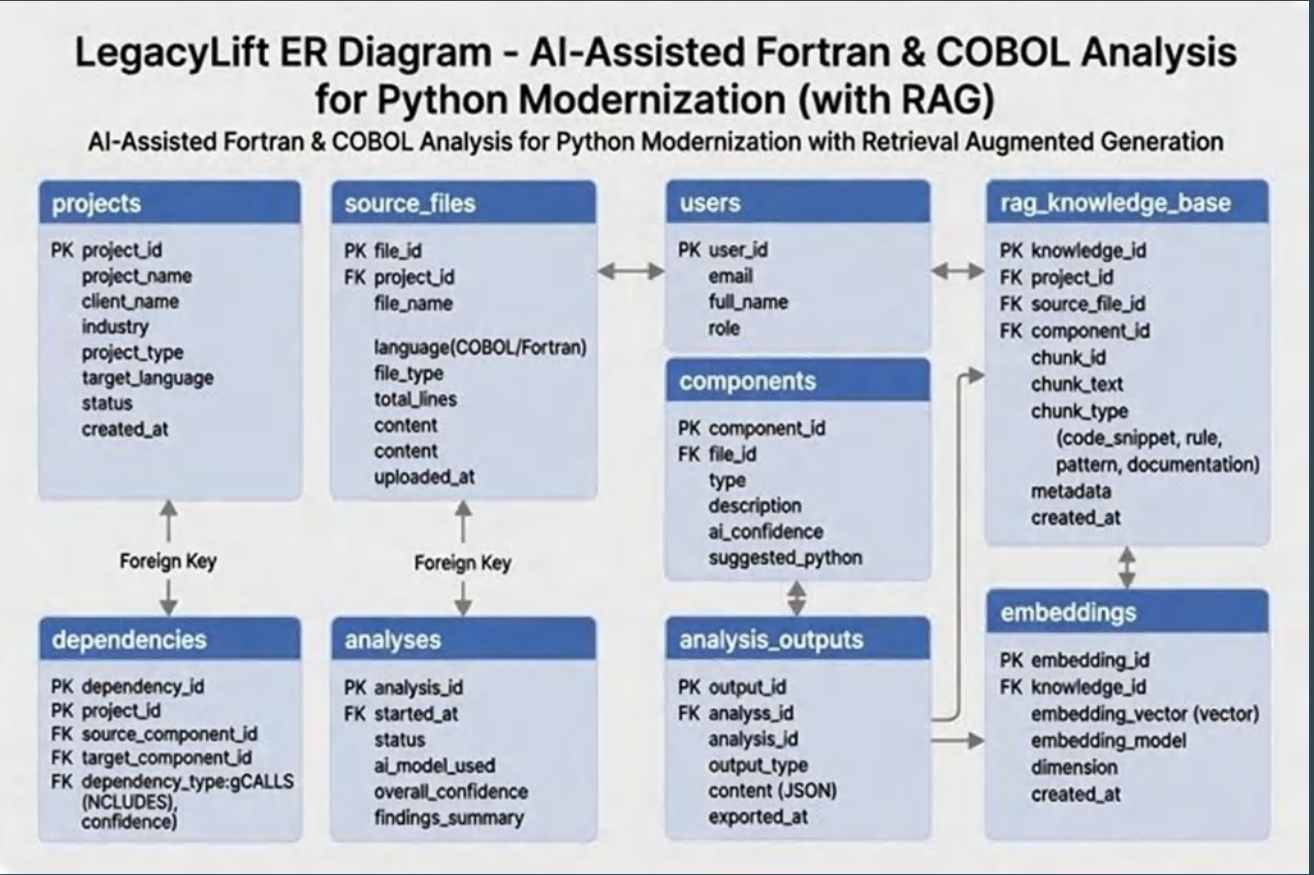
- Legacy code and documentation are split into searchable chunks
- Relevant context is retrieved based on the current task
- Retrieved data is added to the prompt
- Gemini generates analysis or translated code utilizing that context



RAG - Why It Matters for Legacy Lift

- Reduce AI errors
  - Prevents the model from guessing or making up logic
  - Uses real code and context
- Preserves Business Logic
  - Legacy systems contain critical rules spread across multiple files
  - RAG ensures the AI sees related code, dependencies, and comments
- Improves Consistency
  - The framework pulls from the same dataset every time as opposed to what is uploaded at each instance.
- Enables Safer Modernization
  - Reduces risk of breaking critical system behavior during conversion

Database Schema



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.

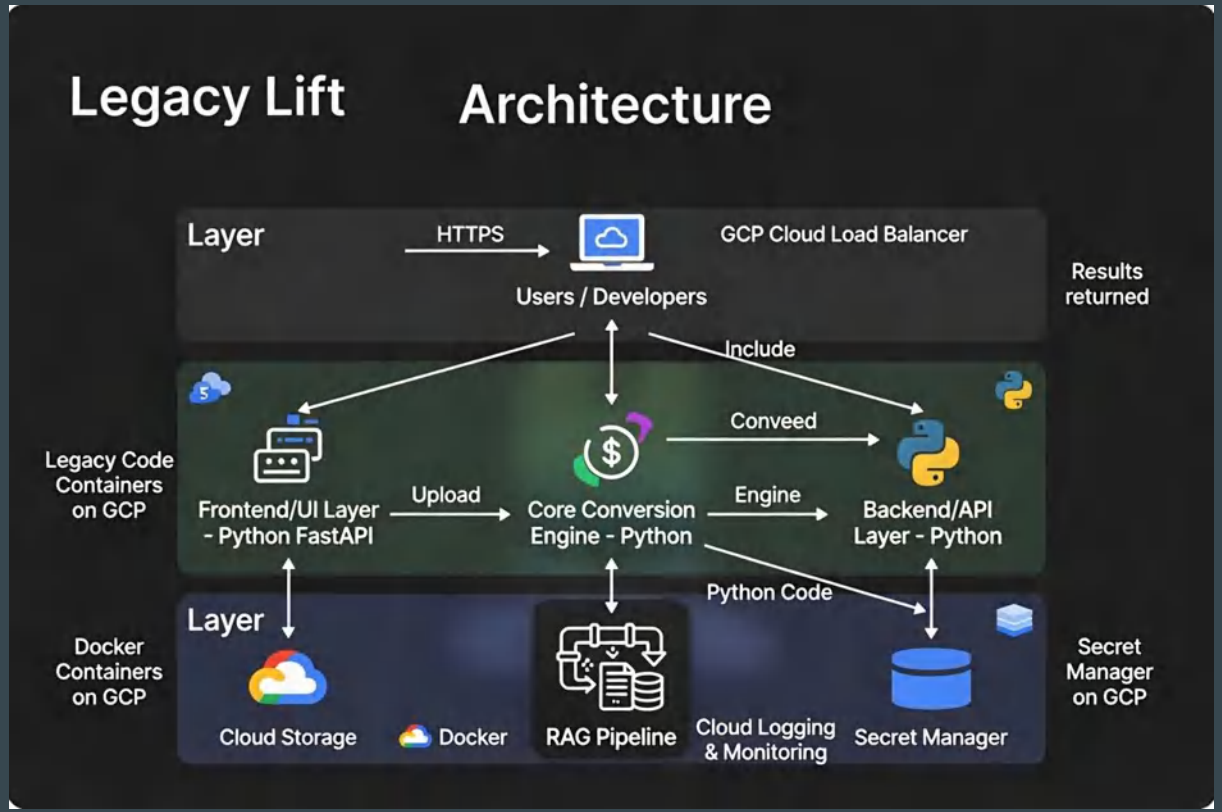
Full Migration

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

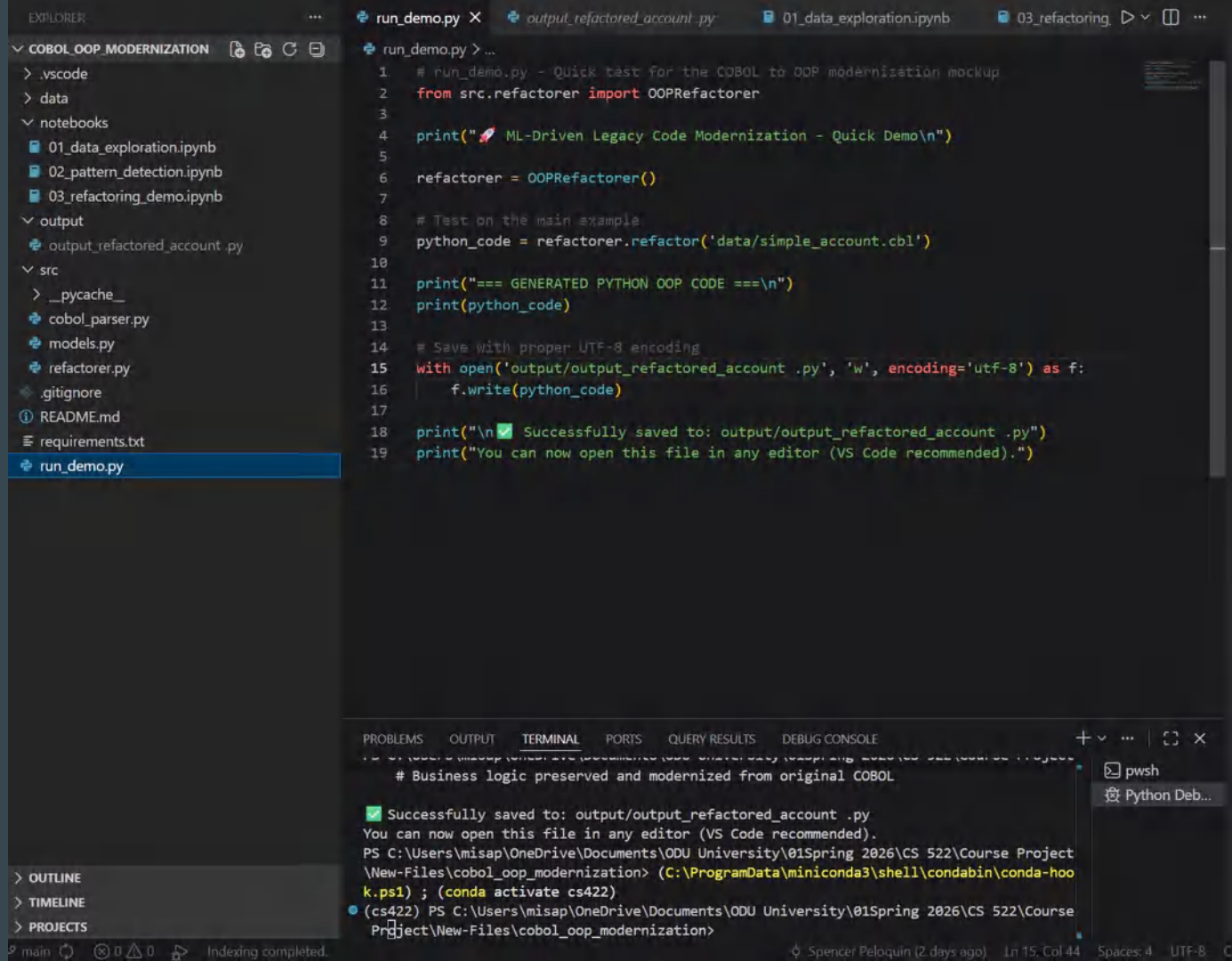
Annual Maintenance

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

Major Functional Components Diagram



Driver


EXPLORER
COBOL_OOP_MODERNIZATION
.vscode
data
notebooks
01_data_exploration.ipynb
02_pattern_detection.ipynb
03_refactoring_demo.ipynb
output
output_refactored_account .py
src
__pycache__
cobol_parser.py
models.py
refactorer.py
.gitignore
README.md
requirements.txt
run_demo.py
run_demo.py
output_refactored_account .py
01_data_exploration.ipynb
03_refactoring
run_demo.py > ...
# run_demo.py - Quick test for the COBOL to OOP modernization mockup
from src.refactorer import OOPRefactorer
print("🚀 ML-Driven Legacy Code Modernization - Quick Demo\n")
refactorer = OOPRefactorer()
# Test on the main example
python_code = refactorer.refactor('data/simple_account.cbl')
print("=== GENERATED PYTHON OOP CODE ===\n")
print(python_code)
# Save with proper UTF-8 encoding
with open('output/output_refactored_account .py', 'w', encoding='utf-8') as f:
f.write(python_code)
print("\n✅ Successfully saved to: output/output_refactored_account .py")
print("You can now open this file in any editor (VS Code recommended).")
PROBLEMS
OUTPUT
TERMINAL
PORTS
QUERY RESULTS
DEBUG CONSOLE
Business logic preserved and modernized from original COBOL
✅ Successfully saved to: output/output_refactored_account .py
You can now open this file in any editor (VS Code recommended).
PS C:\Users\misap\OneDrive\Documents\ODU University\01Spring 2026\CS 522\Course Project\New-Files\cobol_oop_modernization> (C:\ProgramData\miniconda3\shell\condabin\conda-hook.ps1) ; (conda activate cs422)
(cs422) PS C:\Users\misap\OneDrive\Documents\ODU University\01Spring 2026\CS 522\Course Project\New-Files\cobol_oop_modernization>
pwsh
Python Deb...
OUTLINE
TIMELINE
PROJECTS
main
Indexing completed.
Spencer Peloquin (2 days ago)
Ln 15, Col 44
Spaces: 4
UTF-8

Implementation

src > cobol_parser.py > parse_cobol

```python
import re
from typing import Dict, List

def parse_cobol(file_path: str) -> Dict:
    with open(file_path, 'r', encoding='utf-8', errors='ignore') as f:
        content = f.read().upper()

    data = {
        'program_id': re.search(r'PROGRAM-ID\.\s*(\w+)', content),
        'data_items': re.findall(r'0[1-9]\s+\w+.*PIC\s+[\w()V]+', content),
        'paragraphs': re.findall(r'(\w+)-?\w*\.\s*(.*?)(?=\w+-?\w*\.|STOP RUN)', content, re.DOTALL)
    }

    # Extract potential class fields from DATA DIVISION
    fields = []
    for item in data['data_items']:
        match = re.search(r'(\w+)\s+PIC\s+(.+)', item)
        if match:
            fields.append({'name': match.group(1), 'pic': match.group(2)})

    # Paragraphs as potential methods
    methods = [p[0] for p in data['paragraphs'] if p[0] not in ['MAIN-LOGIC', 'STOP']]

    return {
        'program_name': data['program_id'].group(1) if data['program_id'] else 'UNKNOWN',
        'potential_attributes': fields,
        'potential_methods': methods,
        'raw_paragraphs': data['paragraphs']
    }
```

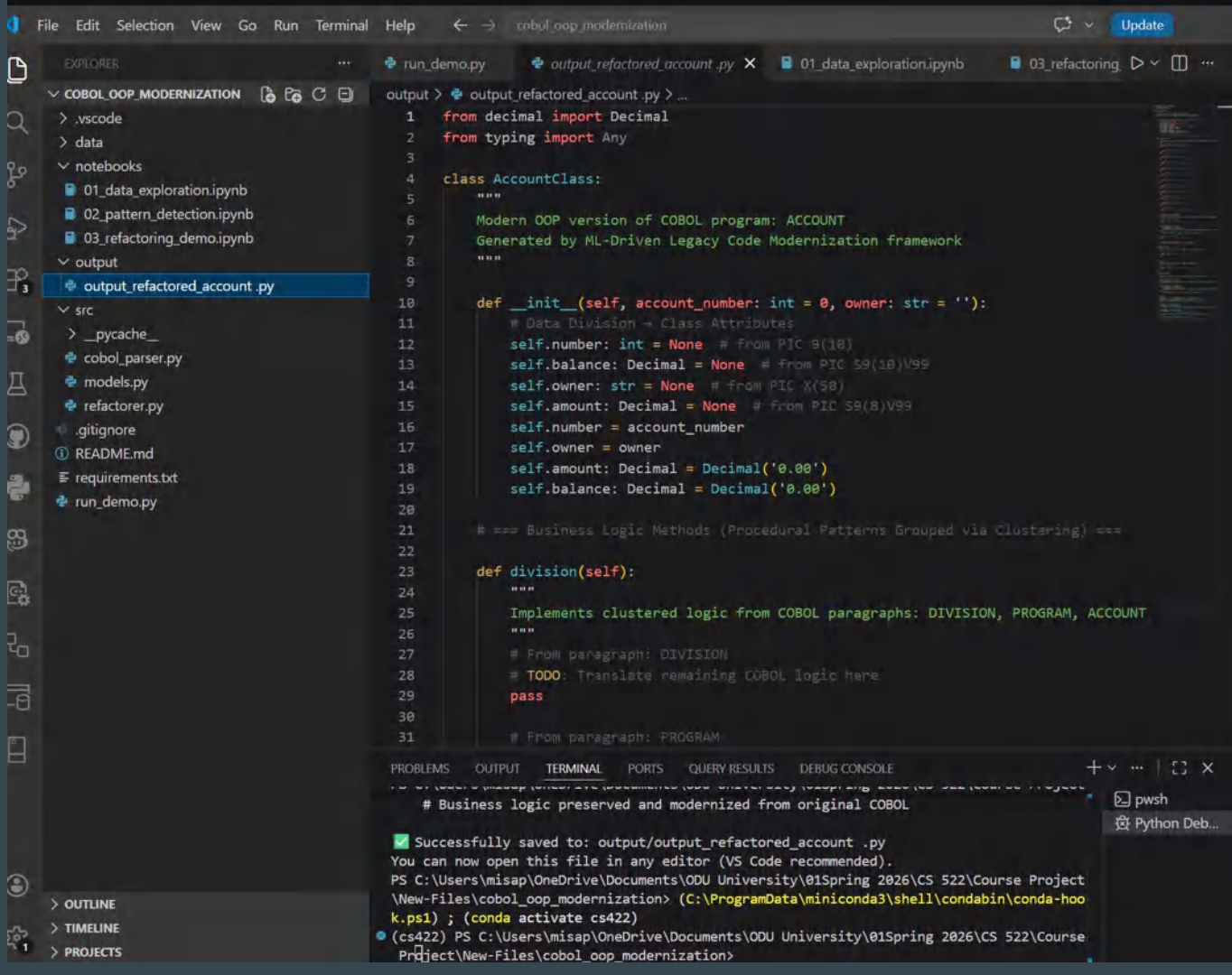
PROBLEMS OUTPUT TERMINAL PORTS QUERY RESULTS DEBUG CONSOLE

```
    # Business logic preserved and modernized from original COBOL

✅ Successfully saved to: output/output_refactored_account .py
You can now open this file in any editor (VS Code recommended).
PS C:\Users\misap\OneDrive\Documents\ODU University\01Spring 2026\CS 522\Course Project\New-Files\cobol_oop_modernization> (C:\ProgramData\miniconda3\shell\condabin\conda-hook.ps1) ; (conda activate cs422)
(cs422) PS C:\Users\misap\OneDrive\Documents\ODU University\01Spring 2026\CS 522\Course Project\New-Files\cobol_oop_modernization>
```

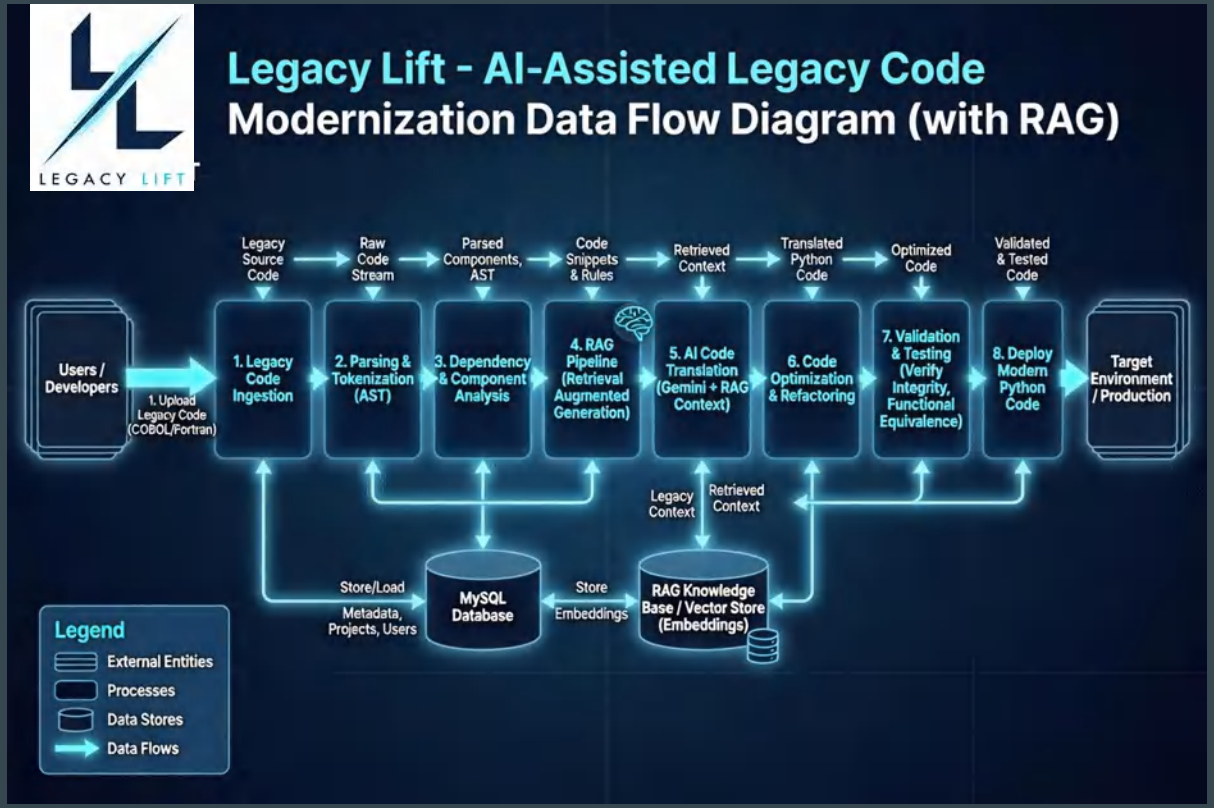
Spencer Peloquin (2 days ago) Ln 29, Col 6 Spaces: 4 UTF-8 CRLF { } MagicPython

Output

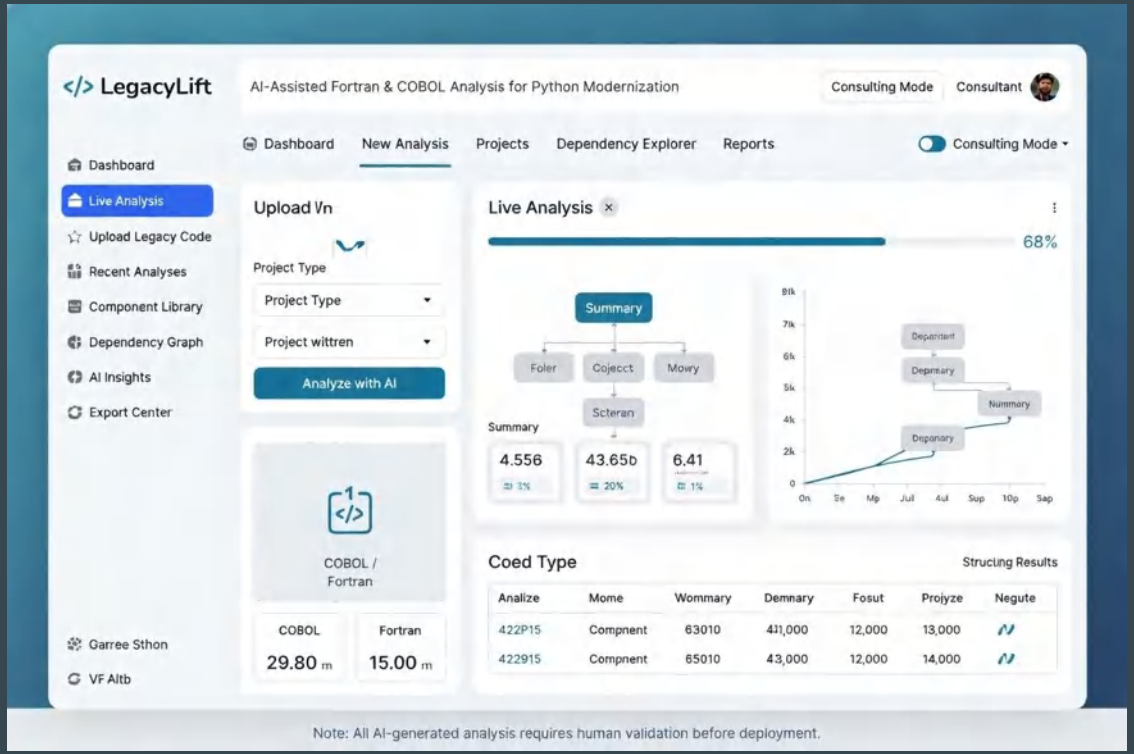

File Edit Selection View Go Run Terminal Help
cobol_oop_modernization
Update
EXPLORER
COBOL_OOP_MODERNIZATION
.vscode
data
notebooks
01_data_exploration.ipynb
02_pattern_detection.ipynb
03_refactoring_demo.ipynb
output
output_refactored_account .py
src
__pycache__
cobol_parser.py
models.py
refactorer.py
.gitignore
README.md
requirements.txt
run_demo.py
run_demo.py
output_refactored_account .py
01_data_exploration.ipynb
03_refactoring
output > output_refactored_account .py > ...
from decimal import Decimal
from typing import Any
class AccountClass:
"""
Modern OOP version of COBOL program: ACCOUNT
Generated by ML-Driven Legacy Code Modernization framework
"""
def __init__(self, account_number: int = 0, owner: str = ''):
# Data Division → Class Attributes
self.number: int = None # from PIC 9(10)
self.balance: Decimal = None # from PIC S9(10)V99
self.owner: str = None # from PIC X(50)
self.amount: Decimal = None # from PIC S9(8)V99
self.number = account_number
self.owner = owner
self.amount: Decimal = Decimal('0.00')
self.balance: Decimal = Decimal('0.00')
# === Business Logic Methods (Procedural Patterns Grouped via Clustering) ===
def division(self):
"""
Implements clustered logic from COBOL paragraphs: DIVISION, PROGRAM, ACCOUNT
"""
From paragraph: DIVISION
TODO: Translate remaining COBOL logic here
pass
From paragraph: PROGRAM
PROBLEMS OUTPUT TERMINAL PORTS QUERY RESULTS DEBUG CONSOLE
Business logic preserved and modernized from original COBOL
Successfully saved to: output/output_refactored_account .py
You can now open this file in any editor (VS Code recommended).
PS C:\Users\misap\OneDrive\Documents\ODU University\01Spring 2026\CS 522\Course Project\New-Files\cobol_oop_modernization> (C:\ProgramData\miniconda3\shell\condabin\conda-hook.ps1) ; (conda activate cs422)
(cs422) PS C:\Users\misap\OneDrive\Documents\ODU University\01Spring 2026\CS 522\Course Project\New-Files\cobol_oop_modernization>
pwsh
Python Deb...
OUTLINE
TIMELINE
PROJECTS

Major Functional Components Diagram

Data Flow Diagram



Project GUI Mockup



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 User Stories - Small Business

As a small business owner, I want to make sure that this transition is seamless and the business can keep running even through the code conversion.

As a small business owner, I want to make sure the conversion happens silently without disruption of day to day operation.

As a small business owner, I want to make sure my employees are just as familiar with the new system as they were with the old system, and there is minimal onboarding.

As a small business owner, I want the implementation time to be as short as possible to avoid disrupting production.

As a small business owner, I want the finished product to be compatible with the same GUI elements as the original product.

As a small business owner, I want to be able to run the existing code during the implementation of the product so that production is not disrupted.

As a small business owner, I want to be able to cancel the implementation and roll back if I no longer want to work with it.

As a small business owner, I want the option for the conversion to be done on premises instead of in the cloud, as I may have hardware or data quirks that would make an upload infeasible.

As a small business, I want the new code to be implemented on day one of completion, as any downtime is costly.

As a customer of this business, I want assurance that my data will be kept secure and not be corrupted.

Appendix User Stories - Production Engineer

As a production engineer, I want the Legacy Lift Python code to have identical functionality to the original COBOL code in system tests.

As a production engineer, I want the legacy Lift to log any changes made during translation process.

As a production engineer, I want the finished product to be able to run on modern hardware.

As a cybersecurity admin, I want the finished product to have no new zero day vulnerabilities.

As a cybersecurity admin, I want the finished product to fix all security flaws known and unpatched in the original product.

As a production engineer, I want the finished product to use fewer resources than the original product.

As a production engineer, I want the finished product to have as few redundancies in production as possible to improve performance.

As a production engineer, I want the framework to have enough tokens purchased so that it does not stall mid-conversion.

As a production engineer, I want there to be an API to embed existing frameworks into LegacyLift.

Appendix User Stories - Enterprise

As a banking firm, I want to make sure that I do not need to rip out and reinstall all of my ATMs just to accommodate the new system.

As an enterprise, I want to make sure that there is some sort of fallback option in case the conversion fails catastrophically.

As a potential enterprise client, I want to make sure that the framework is completely scalable, from a sole accountant's work to a local or regional bank.

As an enterprise, I want the finished product to be compatible with the existing enterprise toolchain so all plugins are compatible.

As an enterprise, I want the finished product to be fully compatible with the existing SQL workflows.

As an enterprise, I want the finished product to integrate with the company's website and backend to avoid a larger conversion.

As an enterprise, I want the conversion to occur in a sandbox to avoid disrupting existing systems.

As an enterprise, I want the AI code to be sanitized to avoid corruption or copyright violations.

As an enterprise, I want existing filesystem hooks to be preserved in order to maintain compatibility.

As an enterprise, I do not want the finished product to sanitize the existing codebase just to make it look prettier, as poorly written code enables functionality that could break if altered.

As an enterprise, I want all of the data sanitized so that no PII is leaked.