FINAL PROJECT TITLE :

Controlled Refactoring Depth Framework: A Proof-of-Concept for Structured Legacy Code Modernization

Conceptual Reference Tool: GPT-5.2 “Controlled Reasoning Depth” architecture (from research study)

Introduction :

Software systems often accumulate technical debt in the form of duplicated logic, deep nesting, global state misuse, and poor modularization. While modern AI-based refactoring systems claim autonomous code improvement, there is limited controlled evaluation of how increasing structural transformation depth impacts measurable code quality metrics. This study proposes a controlled three-level refactoring framework to simulate reasoning depth and measure its impact on legacy Python systems.

Problem Statement :

Legacy Python systems often accumulate structural inefficiencies such as deep nesting, duplicated logic, global state misuse, and poor modularization. While large language models promise autonomous refactoring, controlled evaluation of reasoning depth versus structural improvement remains underexplored. This project proposes a controlled, rule-based refactoring framework that simulates varying “reasoning depths” to measure their impact on code quality metrics in legacy Python modules.

Objectives :

• Design a controlled refactoring framework with three depth levels
• Apply transformations to synthetic legacy modules
• Measure structural and maintainability metrics
• Evaluate quantitative performance trends

Selected Tool :

• Python AST (Abstract Syntax Tree) module
• Radon (cyclomatic complexity analysis)
• Pylint (code quality scoring)
• Static transformation pipeline

Dataset Description :

Synthetic legacy dataset consisting of:

• 8 independent Python modules
• 320–450 lines per module
• ~3,000+ total lines
• High duplication
• Deep nesting
• Debug statements
• No type hints
• Mixed I/O and logic

Stability Engineering:

• Implemented syntax_sanitizer.py to fix incomplete blocks
• Ensured AST compatibility
• Verified compilation using py_compile

Automation:

• Developed run_full_experiment.py
• Processed all 8 modules automatically
• Generated 32 total experiment cases:

- 8 Baseline
- 8 Instant
- 8 Medium
- 8 High

Exported results into experiments/results.csv

This completed the full core experimentation pipeline.

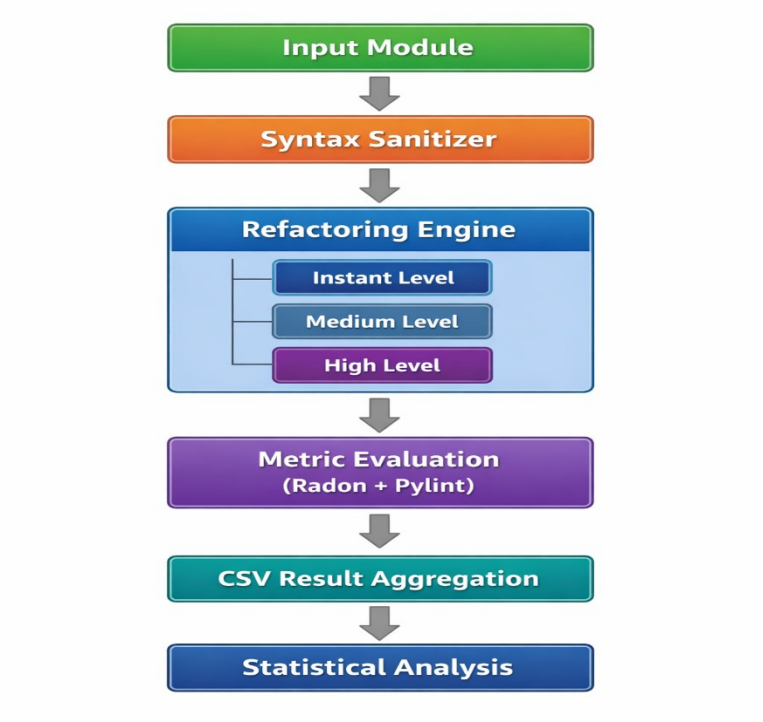

Fig.1. System Architecture

Refactoring Levels :

Level 1 Implementation (Instant Mode):

• Built AST transformer removing print() statements
• Preserved structural integrity using block checks
• Generated refactored output
• Verified syntax correctness

Baseline vs Instant comparison performed successfully.

Level 2 (Medium Mode):

• Implemented automatic docstring injection
• Ensured functions without documentation received structured docstrings
• Maintained functional correctness

Level 3 (High Mode):

• Implemented function splitting for large functions
• Introduced helper functions
• Preserved parameter passing
• Maintained AST correctness

Evaluation Metrics :

• Lines of Code
• Cyclomatic Complexity (Radon)
• Pylint Maintainability Score

Experimental Results :

Average Results Across 8 Modules:

Baseline
Lines: 399.0
Complexity: 125.875
Pylint: 8.885

Instant
Lines: 390.875
Complexity: 125.875
Pylint: 8.8275

Medium
Lines: 444.25
Complexity: 125.875
Pylint: 9.6725

High
Lines: 420.125
Complexity: 125.875
Pylint: 8.91625

Analysis :

• Cosmetic cleanup provides minimal improvement.
• Documentation injection significantly improves maintainability.
• Structural function splitting alone does not reduce algorithmic complexity.
• Cyclomatic complexity reduction requires logical simplification beyond modularization.

Conclusion :

The controlled refactoring depth framework demonstrates that maintainability improvements scale with documentation and structural clarity. However, algorithmic complexity remains unaffected by modular decomposition alone. Effective complexity reduction requires transformation of decision structures rather than structural partitioning.

Limitations :

• No control-flow simplification implemented
• No runtime performance benchmarking
• Complexity limited to cyclomatic metric

Future Work :

• Implement decision simplification transformations
• Add cognitive complexity measurement
• Integrate AI-based logic optimization
• Apply framework to real-world open-source projects