**Request for Proposal (RFP)**

**LLM-Powered Legacy C to Object-Oriented Python Migration Engine**  
*Issued: April 29, 2025*  
*Submission Deadline: May 20, 2025*

**1. Project Overview**

**Objective**

Develop an AI-powered system that automatically converts procedural C code to object-oriented Python by:

1. Parsing C code into a graph-based representation
2. Using Large Language Models to redesign the architecture
3. Generating modern Pythonic object-oriented code

**Background**

Legacy C systems often lack modularity and maintainability. This project aims to create a tool similar to commercial offerings like Macrosoft's MigrateTo.NET2 but with an emphasis on graph-based representation for better logical preservation and custom C++ performance optimizations.

**2. Scope of Work**

**Phase 1: Code Analysis & Graph Generation**

| **Component** | **Requirements** |
| --- | --- |
| Custom C Parser | Develop either a custom parser or integrate with existing tools (pycparser) |
| Logic Graph | Create nodes for functions, variables, and control structures with edges for calls/data flow |
| C++ Performance Layer | Implement performance-critical parsing components in C++ with Python bindings |

**Phase 2: LLM-Based Transformation**

| **Component** | **Requirements** |
| --- | --- |
| Agent Architecture | Implement an agent framework similar to SICA[4](https://github.com/tmgthb/Autonomous-Agents) or LlamaIndex[5](https://github.com/dylanhogg/awesome-python) |
| Prompt Engineering | Design system prompts that instruct migration to OOP patterns |
| Code Generation | Produce Python code with classes, inheritance, type hints, and modern idioms |

**Phase 3: Integration & Testing**

| **Component** | **Requirements** |
| --- | --- |
| C++/Python Bridge | Utilize pybind11 or cppyy[3](https://stackoverflow.com/questions/145270/calling-c-c-from-python) for high-performance components |
| Testing Framework | Generate and execute tests to verify functional equivalence |
| Self-Improvement | Implement feedback loop for iterative refinement |

**3. Technical Requirements**

**C/C++ Integration Options**

Vendors should propose one of the following approaches:

* **Option A**: Custom C++ parser with pybind11[3](https://stackoverflow.com/questions/145270/calling-c-c-from-python) bindings
* **Option B**: Existing parser with cppyy[3](https://stackoverflow.com/questions/145270/calling-c-c-from-python) for runtime bindings
* **Option C**: SWIG interface for complex C++ to Python translation[3](https://stackoverflow.com/questions/145270/calling-c-c-from-python)

**Agent Framework Requirements**

* Support for autonomous decision-making during code transformation
* Ability to handle multi-file projects with interdependencies
* Self-reflection capabilities for code quality improvement

**Performance Targets**

* Process minimum 10,000 LOC per hour
* Achieve >80% automated translation rate
* Maintain <5% performance degradation between C and Python versions

**4. Deliverables**

| **Item** | **Description** | **Due Date** |
| --- | --- | --- |
| Parser & Graph Generator | C/C++ engine with Python interface | Week 8 |
| LLM Agent System | Based on frameworks like CrewAI or LlamaIndex[5](https://github.com/dylanhogg/awesome-python) | Week 14 |
| Testing Framework | Automated validation suite | Week 18 |
| Documentation | Technical specs and user guide | Week 20 |
| Source Code | Complete codebase with build scripts | Week 24 |

**5. Technical Approach**

Proposals should describe approach to:

1. **Parser Development**:
   * Custom development vs. existing tools
   * Error handling and recovery strategies
   * C preprocessor directive management
2. **C++ Integration**:
   * Method for binding C++ to Python (pybind11, cppyy, SWIG)[3](https://stackoverflow.com/questions/145270/calling-c-c-from-python)
   * Performance optimization strategy
   * Memory management
3. **LLM Agent Design**:
   * Framework selection (from options such as LlamaIndex, CrewAI, etc.)[5](https://github.com/dylanhogg/awesome-python)
   * Prompt design methodology
   * Self-improvement capabilities[4](https://github.com/tmgthb/Autonomous-Agents)

**6. Budget & Timeline**

| **Phase** | **Timeline** | **Budget Range** |
| --- | --- | --- |
| Planning & Design | 4 weeks | $25K-$35K |
| Parser Development | 8 weeks | $50K-$70K |
| Agent Implementation | 8 weeks | $50K-$70K |
| Testing & Refinement | 4 weeks | $30K-$40K |
| **Total** | **24 weeks** | **$155K-$215K** |

**7. Evaluation Criteria**

| **Criteria** | **Weight** | **Description** |
| --- | --- | --- |
| Technical Approach | 35% | Soundness of parsing and LLM integration strategy |
| Performance | 25% | Speed and accuracy of code migration |
| Team Experience | 20% | Prior work with C/C++, Python, and LLMs |
| Cost | 10% | Value for investment |
| Timeline | 10% | Delivery schedule feasibility |

**8. Submission Guidelines**

Proposals must include:

1. Technical approach with architecture diagram
2. Team composition and relevant experience
3. Project timeline with milestones
4. Cost breakdown
5. Sample implementation (small-scale demo)

Submit to: code-migration-rfp@company.com by May 20, 2025.

**9. Appendices**

**Appendix A: Sample Migration**

**Input (C):**

c

**struct** point {

**float** x;

**float** y;

};

**void** translate(**struct** point \*p, **float** dx, **float** dy) {

p->x += dx;

p->y += dy;

}

**Output (Python):**

python

**class** Point:

**def** \_\_init\_\_(self, x: float = 0.0, y: float = 0.0):

self.x = x

self.y = y

**def** translate(self, dx: float, dy: float) -> None:

self.x += dx

self.y += dy

**Appendix B: Reference Projects**

* SICA: Self-Improving Coding Agent[4](https://github.com/tmgthb/Autonomous-Agents)
* Macrosoft MigrateTo.NET service2
* LLM-Assisted Translation of Legacy FORTRAN Codes to C++