Python refactoring to C++:

1. Understand the Python Code**:** Before refactoring, make sure you understand the logic and functionality of the Python code. This will help you in the translation process.
2. Type Declarations: C++ is statically-typed, so explicit type declarations are necessary. Ensure that all variables have their types explicitly mentioned.
3. Containers and Collections: Translate Python lists, dictionaries, and sets into their C++ equivalents. For instance, replace Python lists with C++ vectors.
4. Functions and Classes: Pay attention to function prototypes and class definitions. In C++, you need to specify the return type of functions, unlike Python.
5. Memory Management: Unlike Python's automatic memory management, C++ requires manual memory allocation and deallocation. Use new and delete or consider smart pointers.
6. Libraries: Replace Python libraries with C++ equivalents, especially if your AI project relies on specific Python libraries like TensorFlow or PyTorch.
7. Error Handling: C++ typically uses exceptions for error handling. Adjust error-handling mechanisms to conform to C++ standards.
8. Compile and Test: After refactoring, compile the C++ code and rigorously test it to ensure correctness. Debug any issues that arise during this process.
9. Optimization: Leverage C++'s potential for low-level optimizations to enhance performance, which might not be as accessible in Python.
10. Documentation: Ensure that the refactored code is well-documented. This is crucial for maintainability, especially when transitioning from a language with different syntax and conventions.
11. Best Practices: Familiarize yourself with C++ best practices and idioms. Consider design patterns and coding standards to write clean and efficient C++ code.