Building Cython and C++ RMI Implementations

1. Cython Setup

setup_cython.py

```python
from setuptools import setup, Extension
from Cython.Build import cythonize
import numpy as np

extensions = [
    Extension(
        "rmi_cython",
        ["rmi_cython.pyx"],
        include_dirs=[np.get_include()],
        extra_compile_args=['-O3', '-march=native'],  # Maximum optimization
    )
]

setup(
    ext_modules=cythonize(extensions,
                          compiler_directives={'language_level': "3"}),
    zip_safe=False,
)
```

Build command:

```bash
# Install dependencies
pip install cython numpy

# Build the extension
python setup_cython.py build_ext --inplace
```

2. C++ Implementation

rmi_cpp.cpp

cpp

```
#include <Python.h>
#include <numpy/arrayobject.h>
#include <vector>
#include <algorithm>
#include <cmath>

class RMI {
private:
    std::vector<double> data;
    double slope;
    double intercept;
    int max_error;
    size_t n;

    void train_model() {
        // Calculate linear regression parameters
        double x_sum = 0.0, y_sum = 0.0, xy_sum = 0.0, xx_sum = 0.0;

        for (size_t i = 0; i < n; ++i) {
            double x = data[i];
            double y = static_cast<double>(i);
            x_sum += x;
            y_sum += y;
            xy_sum += x * y;
            xx_sum += x * x;
        }

        double x_mean = x_sum / n;
        double y_mean = y_sum / n;
        double denominator = xx_sum - x_sum * x_mean;

        if (denominator != 0) {
            slope = (xy_sum - x_sum * y_mean) / denominator;
            intercept = y_mean - slope * x_mean;
        } else {
            slope = 0.0;
            intercept = y_mean;
        }

        // Calculate max error
        double max_err = 0.0;
        for (size_t i = 0; i < n; ++i) {
            double pred = slope * data[i] + intercept;
            double error = std::abs(pred - i);
            max_err = std::max(max_err, error);
        }
        max_error = static_cast<int>(max_err + 1);
    }

    int binary_search(double key, int left, int right) const {
        while (left <= right) {
            int mid = (left + right) / 2;
            if (data[mid] == key) return mid;
            if (data[mid] < key) left = mid + 1;
            else right = mid - 1;
        }
        return -1;
    }

public:
    RMI(const double* keys, size_t size) : n(size) {
        data.reserve(n);
        for (size_t i = 0; i < n; ++i) {
            data.push_back(keys[i]);
        }
        train_model();
    }

    int lookup(double key) const {
        // Model prediction
        int pos = static_cast<int>(slope * key + intercept);
        pos = std::max(0, std::min(pos, static_cast<int>(n - 1)));

        // Quick check
        if (data[pos] == key) return pos;

        // Binary search with bounds
        int left = std::max(0, pos - max_error);
        int right = std::min(static_cast<int>(n - 1), pos + max_error);

        return binary_search(key, left, right);
    }

    double get_slope() const { return slope; }
    double get_intercept() const { return intercept; }
    int get_max_error() const { return max_error; }
};

// Python wrapper
typedef struct {
    PyObject_HEAD
    RMI* rmi;
} PyRMI;

static void PyRMI_dealloc(PyRMI* self) {
    delete self->rmi;
    Py_TYPE(self)->tp_free((PyObject*)self);
}

static PyObject* PyRMI_new(PyTypeObject* type, PyObject* args, PyObject* kwds) {
    PyRMI* self = (PyRMI*)type->tp_alloc(type, 0);
    return (PyObject*)self;
}

static int PyRMI_init(PyRMI* self, PyObject* args, PyObject* kwds) {
    PyObject* keys_obj;

    if (!PyArg_ParseTuple(args, "O", &keys_obj)) {
        return -1;
    }

    // Convert to numpy array
    PyArrayObject* keys_array = (PyArrayObject*)PyArray_FROM_OTF(
        keys_obj, NPY_DOUBLE, NPY_ARRAY_IN_ARRAY);

    if (keys_array == NULL) {
        return -1;
    }

    npy_intp size = PyArray_SIZE(keys_array);
    double* data = (double*)PyArray_DATA(keys_array);

    self->rmi = new RMI(data, size);

    Py_DECREF(keys_array);
    return 0;
}
```

```
static PyObject* PyRMI_lookup(PyRMI* self, PyObject* args) {
    double key;

    if (!PyArg_ParseTuple(args, "d", &key)) {
        return NULL;
    }

    int result = self->rmi->lookup(key);
    return PyLong_FromLong(result);
}

static PyMethodDef PyRMI_methods[] = {
    {"lookup", (PyCFunction)PyRMI_lookup, METH_VARARGS, "Lookup a key"},
    {NULL}  // Sentinel
};

static PyTypeObject PyRMIType = {
    PyVarObject_HEAD_INIT(NULL, 0)
    .tp_name = "rmi_cpp.RMI",
    .tp_doc = "C++ RMI implementation",
    .tp_basicsize = sizeof(PyRMI),
    .tp_itemsize = 0,
    .tp_flags = Py_TPFLAGS_DEFAULT | Py_TPFLAGS_BASETYPE,
    .tp_new = PyRMI_new,
    .tp_init = (initproc)PyRMI_init,
    .tp_dealloc = (destructor)PyRMI_dealloc,
    .tp_methods = PyRMI_methods,
};

static PyModuleDef rmi_cpp_module = {
    PyModuleDef_HEAD_INIT,
    .m_name = "rmi_cpp",
    .m_doc = "C++ RMI module",
    .m_size = -1,
};

PyMODINIT_FUNC PyInit_rmi_cpp(void) {
    import_array();  // Initialize numpy

    if (PyType_Ready(&PyRMIType) < 0) {
        return NULL;
    }

    PyObject* m = PyModule_Create(&rmi_cpp_module);
    if (m == NULL) {
        return NULL;
    }

    Py_INCREF(&PyRMIType);
    if (PyModule_AddObject(m, "RMI", (PyObject*)&PyRMIType) < 0) {
        Py_DECREF(&PyRMIType);
        Py_DECREF(m);
        return NULL;
    }

    return m;
}
```

setup_cpp.py

python

```
from setuptools import setup, Extension
import numpy as np

cpp_extension = Extension(
    'rmi_cpp',
    sources=['rmi_cpp.cpp'],
    include_dirs=[np.get_include()],
    extra_compile_args=['-O3', '-std=c++11', '-march=native'],
    language='c++'
)

setup(
    name='rmi_cpp',
    ext_modules=[cpp_extension],
    zip_safe=False,
)
```

Build command:

bash

```
python setup_cpp.py build_ext --inplace
```

3. Build Script

build_all.sh

```bash
#!/bin/bash

echo "Building Cython RMI..."
python setup_cython.py build_ext --inplace

echo "Building C++ RMI..."
python setup_cpp.py build_ext --inplace

echo "Done! You can now run: python test_fast_rmi.py"
```

Make it executable:

```bash
chmod +x build_all.sh
./build_all.sh
```