AI Model Marketplace - Complete Guide

1. Code Architecture

Directory Structure

```
.
├── src/                # Core C++ source files
│   ├── agent.cpp      # Intelligent agent implementation
│   ├── agent.hpp      # Agent interface definitions
│   ├── model.cpp      # AI model implementation
│   ├── model.hpp      # Model interface definitions
│   ├── blockchain.cpp # Mock blockchain functionality
│   ├── blockchain.hpp # Blockchain interface
│   ├── storage.cpp    # Model storage implementation
│   ├── storage.hpp    # Storage interface
│   ├── utils.cpp      # Utility functions
│   └── utils.hpp      # Utility declarations
├── models/            # Directory for stored models
└── Makefile          # Build configuration
```

Core Components

1. AI Model System (model.hpp/cpp)

- Handles multiple media types (text, image, audio, video)
- Manages model weights and versions
- Provides training and inference capabilities
- Example:

```
// Create a multi-modal model
std::vector<MediaType> types = {MediaType::TEXT, MediaType::IMAGE};
AIModel model("MyModel", types);

// Train with different media
model.trainWithText("Sample text");
model.trainWithImage(imageData);
```

2. Agent System (agent.hpp/cpp)

- Provides intelligent decision-making
- Manages model operations
- Handles context and state
- Example:

```
// Create an agent for a model
auto model = std::make_shared<AIModel>("AgentModel", types);
ModelAgent agent(model);

// Process with context
AgentContext context{
    .mediaType = MediaType::TEXT,
    .input = "Sample",
    .parameters = {{"mode", "analysis"}}
};
agent.processContext(context);
```

3. Blockchain System (blockchain.hpp/cpp)

- Handles transactions and model ownership
- Manages rental agreements
- Tracks resource contributions
- Example:

```
BlockchainLedger ledger;
ledger.addTransaction("RENT", modelId, renter, owner, price);
ledger.updateResourceContribution(contributor, gpuHours);
```

2. Building and Running

Building from Source

```
# Clean previous builds
make clean

# Build the application
make

# Run the application
./aimarket
```

Running Tests

```
# Run all tests
./aimarket --test

# Run specific test categories
./aimarket --test-media
./aimarket --test-agent
./aimarket --test-blockchain
```

3. Training Models

Basic Training

```
// Create model
std::vector<MediaType> types = {MediaType::TEXT};
AIModel model("TextModel", types);

// Train with text
model.trainWithText("Training data");
std::cout << "Accuracy: " << model.getAccuracy() << std::endl;
```

Advanced Training with Agent

```
// Setup agent
auto model = std::make_shared<AIModel>("SmartModel", types);
ModelAgent agent(model);

// Training context
AgentContext context{
    .mediaType = MediaType::TEXT,
    .input = "Training data",
    .parameters = {
        {"mode", "training"},
        {"iterations", "10"},
        {"learning_rate", "0.01"}
    }
};

// Train with feedback
agent.processContext(context);
agent.learn(context, "Good performance");
```

4. Market Operations

Renting Models

```
// As a model owner
BlockchainLedger ledger;
const double rentalPrice = 10.0;
const time_t duration = 24 * 3600; // 24 hours

// List model for rent
ledger.addTransaction("RENT_LISTING", modelId, owner, "", rentalPrice);

// As a renter
ledger.addTransaction("RENT", modelId, renter, owner, rentalPrice, duration);

// Check rental status
if (ledger.isModelRentedBy(modelId, renter)) {
    // Use the rented model
}
```

Selling Models

```
// Calculate fair price
double fairPrice = ledger.calculateFairPrice(modelId);

// List model for sale
ledger.addTransaction("SALE_LISTING", modelId, owner, "", fairPrice);

// Purchase model
ledger.addTransaction("PURCHASE", modelId, buyer, owner, fairPrice);
```

5. Resource Management

Memory Considerations

- Text models: ~25.6MB
- Image models: ~176.1MB
- Audio models: ~184.3MB
- Video models: ~405.2MB

```
// Check memory requirements
const auto& props = model.getMediaProperties(MediaType::VIDEO);
size_t requiredMemory = props.visual.width * props.visual.height *
                        props.visual.channels * props.visual.frameRate;
```

Resource Contribution

```
// Contribute computing resources
ledger.updateResourceContribution(contributor, gpuHours);

// Get contributor reputation
auto reputation = ledger.getUserReputation(contributor);
std::cout << "Reputation score: " << reputation.score << std::endl;
```

6. Advanced Features

Quality Control

```
// Set quality metrics
QualityMetrics metrics{
    .accuracy = 0.95,
    .reliability = 0.98,
    .userCount = 100,
    .avgResponseTime = 0.05
};
ledger.updateQualityMetrics(modelId, metrics);

// Validate model
ledger.validateModel(modelId, "validator1");
```

Collaborative Training

```
// Record collaborative session
std::vector<std::string> contributors = {"alice", "bob"};
std::vector<double> contributions = {10.5, 8.3}; // GPU hours
ledger.addCollaborativeTransaction(modelId, contributors, contributions);
```

Version Control

```
// Save version
model.save();

// Load specific version
model.load(modelId);

// Track version history
auto history = ledger.getVersionHistory(modelId);
```

7. Best Practices

1. Memory Management

```
// Monitor memory usage
for (const auto& type : model.getSupportedTypes()) {
    const auto& props = model.getMediaProperties(type);
    std::cout << "Memory for " << static_cast<int>(type)
              << ": " << props.inputSize * props.outputSize << " bytes\n";
}
```

2. Error Handling

```
try {
    agent.processContext(context);
} catch (const std::exception& e) {
    std::cerr << "Error: " << e.what() << std::endl;
    agent.setState(AgentState::ERROR);
}
```

3. Resource Cleanup

```
// Clear history after processing
if (agent.getState() != AgentState::ERROR) {
    model->clearWeightHistory();
}
```

8. Troubleshooting

Common Issues

1. Memory Errors

```
// Check available memory
const auto& props = model->getMediaProperties(MediaType::VIDEO);
if (props.inputSize * props.outputSize > availableMemory) {
    throw std::runtime_error("Insufficient memory");
}
```

2. Training Issues

```
// Monitor training progress
double previousAccuracy = model->getAccuracy();
model->train();
if (model->getAccuracy() <= previousAccuracy) {
    std::cout << "Warning: No improvement in accuracy" << std::endl;
}
```

3. Blockchain Errors

```
// Verify transaction
if (!ledger.verifyTransaction(transactionId)) {
    std::cerr << "Invalid transaction" << std::endl;
}
```

9. Contributing

Adding New Features

1. Follow the existing code structure
2. Add appropriate tests
3. Update documentation
4. Verify memory management
5. Test with the agent system

Code Style

- Use consistent naming conventions
- Add comments for complex logic
- Include debug information
- Follow C++17 best practices

10. Future Extensions

Planned Features

1. Real blockchain integration
2. Advanced model training
3. GUI implementation
4. Real-time processing

Integration Points

- Blockchain interface in `blockchain.hpp`
- Model extension points in `model.hpp`

- Agent customization in `agent.hpp`