Clean any previous builds

make clean

Build the application

make

Verify the build

./aimarket --version

```

2. Directory structure:
```

.
├── models/ # Trained model storage
├── src/ # Source code
└── aimarket # Main executable

## Using the Application

### 1. Basic Operations

#### Starting the Application
```bash
./aimarket
```

Running Tests

```
./aimarket --test
```

2. Training Models

Creating a New Model

```
// Create a model with desired media types
std::vector<MediaType> types = {
    MediaType::TEXT,    // For text processing
    MediaType::IMAGE    // For image processing
};
auto model = std::make_shared<AIModel>("MyModel", types);
ModelAgent agent(model);
```

Training with Different Media Types

1. Text Training:

```
AgentContext textContext{
    .mediaType = MediaType::TEXT,
    .input = "Sample training text",
    .parameters = {{"mode", "training"}}
};
agent.processContext(textContext);

// Check training progress
std::cout << "Model accuracy: " << model->getAccuracy() << std::endl;
std::cout << "Model version: " << model->getVersion() << std::endl;
```

2. Image Training:

```
// Load image data
std::vector<uint8_t> imageData = utils::loadBinaryFile("training_image.jpg");

// Create training context
AgentContext imageContext{
    .mediaType = MediaType::IMAGE,
    .binaryData = imageData,
    .parameters = {{"mode", "training"}}
};
agent.processContext(imageContext);
```

3. Audio Training:

```
std::vector<uint8_t> audioData = utils::loadBinaryFile("training_audio.wav");
AgentContext audioContext{
    .mediaType = MediaType::AUDIO,
    .binaryData = audioData,
    .parameters = {{"mode", "training"}}
};
agent.processContext(audioContext);
```

4. Video Training:

```
std::vector<uint8_t> videoData = utils::loadBinaryFile("training_video.mp4");
AgentContext videoContext{
    .mediaType = MediaType::VIDEO,
    .binaryData = videoData,
    .parameters = {{"mode", "training"}}
};
agent.processContext(videoContext);
```

3. Processing Content

Text Processing

```
AgentContext context{
    .mediaType = MediaType::TEXT,
    .input = "Text to process",
    .parameters = {{"mode", "process"}}
};
agent.processContext(context);

// Get agent's reasoning
std::cout << agent.getActionReasoning() << std::endl;
```

Image Processing

```
std::vector<uint8_t> imageData = utils::loadBinaryFile("image.jpg");
AgentContext context{
    .mediaType = MediaType::IMAGE,
    .binaryData = imageData,
    .parameters = {{"mode", "process"}}
};
agent.processContext(context);
```

4. Agent Learning and Feedback

Providing Feedback

```
// After processing, provide feedback
agent.learn(context, "Excellent performance on text analysis");
agent.learn(context, "Image processing needs improvement");
```

Monitoring Agent State

```
// Check agent state
AgentState state = agent.getState();
if (state == AgentState::ERROR) {
    std::cout << "Error occurred: " << agent.getActionReasoning() << std::endl;
}
```

Best Practices

1. Memory Management

- Monitor memory usage when processing large files
- Clean up resources after processing
- Use appropriate batch sizes for training

2. Error Handling

```
try {
    // Validate context before processing
    if (!agent.validateContext(context)) {
        throw std::runtime_error("Invalid context");
    }
    agent.processContext(context);
} catch (const std::exception& e) {
    std::cerr << "Error: " << e.what() << std::endl;
    agent.setState(AgentState::ERROR);
}
```

3. Performance Optimization

- Process large files in chunks
- Monitor system memory usage
- Implement proper cleanup routines
- Use efficient data formats for each media type

Troubleshooting

Common Issues and Solutions

1. **Compilation Errors**

```
# Clean and rebuild
make clean
make
```

2. **Memory Errors**

```
// Check available memory before processing large files
const auto& props = model->getMediaProperties(MediaType::VIDEO);
size_t requiredMemory = props.visual.width * props.visual.height *
                       props.visual.channels * props.visual.frameRate;
```

3. **Model Loading Errors**

```
try {
    model->load(modelId);
} catch (const std::exception& e) {
    std::cerr << "Failed to load model: " << e.what() << std::endl;
}
```

Support and Documentation

For additional help:

1. Check the API documentation in `documentation.md`
2. Review error logs in the console output
3. Monitor agent state and reasoning
4. Check model validation status

Command Reference

Basic Commands

```
# Build
make clean && make

# Run
./aimarket

# Run tests
./aimarket --test

# Version
./aimarket --version

# Help
./aimarket --help
```

Environment Setup

```
# Create models directory
mkdir -p models

# Set permissions
chmod +x aimarket
```

Appendix

A. Media Type Support

```
enum class MediaType {
    TEXT = 1,
    IMAGE = 2,
    AUDIO = 4,
    VIDEO = 8
};
```

B. Agent States

```
enum class AgentState {
    IDLE,
    PROCESSING,
    TRAINING,
    ERROR
};
```

C. Agent Actions

```
enum class AgentAction {
    ANALYZE,
    TRAIN,
    PROCESS,
    WAIT
};
```