// Text model properties

- Maximum sequence length: 1024 tokens
- Vocabulary size: 50,000 tokens
- Weight size: ~25.6MB

2. Image Processing
```cpp
// Image model properties
- Resolution: 224x224 pixels
- Channels: 3 (RGB)
- Weight size: ~176.1MB
```

3. Audio Processing

```
// Audio model properties
- Sample rate: 16kHz
- Channels: 1 (mono)
- Bit depth: 16-bit
- Weight size: ~184.3MB
```

4. Video Processing

```
// Video model properties
- Resolution: 224x224 pixels
- Frame rate: 30 fps
- Channels: 3 (RGB)
- Weight size: ~405.2MB
```

Intelligent Agent System

Agent Capabilities

1. Autonomous Decision Making
    - Context-aware processing
    - Dynamic action selection
    - Performance optimization

2. Multi-Modal Support
    - Unified interface for all media types
    - Automatic media type detection
    - Optimized processing pipelines

3. Learning and Adaptation
    - Feedback-based learning
    - Performance tracking
    - Behavior optimization

4. State Management

  - Process monitoring
  - Error handling
  - Recovery mechanisms

Agent States

```
enum class AgentState {
    IDLE,       // Ready for new tasks
    PROCESSING, // Currently handling a task
    TRAINING,   // Updating model
    ERROR       // Error state
};
```

Agent Actions

```
enum class AgentAction {
    ANALYZE,  // Analyze input data
    TRAIN,    // Train model
    PROCESS,  // Process input
    WAIT      // Wait for better context
};
```

Usage Examples

1. Creating a Multi-Modal Agent

```
// Initialize model with all media types
std::vector<MediaType> allTypes = {
    MediaType::TEXT,
    MediaType::IMAGE,
    MediaType::AUDIO,
    MediaType::VIDEO
};
auto model = std::make_shared<AIModel>("Universal-Model", allTypes);
ModelAgent agent(model);
```

2. Processing Different Media Types

```cpp
// Text processing
AgentContext textContext{
    .mediaType = MediaType::TEXT,
    .input = "Sample text input",
    .parameters = {{"mode", "analysis"}}
};
agent.processContext(textContext);

// Image processing
std::vector<uint8_t> imageData = loadImageData();
AgentContext imageContext{
    .mediaType = MediaType::IMAGE,
    .binaryData = imageData,
    .parameters = {{"mode", "processing"}}
};
agent.processContext(imageContext);
```

3. Training and Learning

```cpp
// Train with feedback
agent.learn(context, "Good performance on text analysis");
std::cout << "Agent reasoning: " << agent.getActionReasoning() << "\n";
```

Best Practices

1. Context Management

```cpp
// Provide complete context information
AgentContext context{
    .mediaType = MediaType::TEXT,
    .input = "Input data",
    .parameters = {
        {"mode", "analysis"},
        {"priority", "high"}
    }
};
```

2. Resource Management

- Monitor memory usage for different media types
- Use appropriate batch sizes
- Handle large files efficiently

3. Error Handling

```
try {
    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);
}
```

API Reference

ModelAgent Class

```
class ModelAgent {
public:
    // Constructor
    ModelAgent(std::shared_ptr<AIModel> model);

    // Core operations
    void processContext(const AgentContext& context);
    AgentAction decideNextAction(const AgentContext& context);
    void executeAction(AgentAction action, const AgentContext& context);

    // State management
    AgentState getState() const;
    void setState(AgentState newState);

    // Learning and adaptation
    void learn(const AgentContext& context, const std::string& feedback);
    std::string getActionReasoning() const;
};
```

AIModel Class

```
class AIModel {
public:
    // Multi-modal support
    bool supportsMediaType(MediaType type) const;
    std::vector<MediaType> getSupportedTypes() const;

    // Media processing
    std::string processText(const std::string& input) const;
    std::vector<uint8_t> processImage(const std::vector<uint8_t>& input) const;
    std::vector<uint8_t> processAudio(const std::vector<uint8_t>& input) const;
    std::vector<uint8_t> processVideo(const std::vector<uint8_t>& input) const;

    // Training methods
    void trainWithText(const std::string& text);
    void trainWithImage(const std::vector<uint8_t>& imageData);
    void trainWithAudio(const std::vector<uint8_t>& audioData);
    void trainWithVideo(const std::vector<uint8_t>& videoData);
};
```

Performance Considerations

Memory Usage

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

Optimization Tips

1. Use appropriate batch sizes for different media types
2. Monitor memory usage during training
3. Implement proper cleanup for large media files
4. Use efficient data formats for each media type

Error Handling

Common Issues

1. Invalid Context

```
if (!agent.validateContext(context)) {
    std::cerr << "Invalid context provided" << std::endl;
    return;
}
```

2. Media Type Mismatch

```
if (!model->supportsMediaType(mediaType)) {
    throw std::runtime_error("Unsupported media type");
}
```

3. Resource Exhaustion

```
try {
    agent.processContext(largeContext);
} catch (const std::bad_alloc& e) {
    std::cerr << "Memory allocation failed" << std::endl;
    agent.setState(AgentState::ERROR);
}
```