ARIA Project Structure

📂 Directory Overview

plaintext

```
aria/
├── aria.py                    # Main application entry point
├── config.py                  # Configuration management
├── llm_interface.py           # Ollama integration
├── plugin_system.py           # Plugin loader and manager
├── learning_system.py         # Learning and feedback system
│
├── tools/                     # Tool system
│   ├── __init__.py
│   ├── base.py               # Tool framework
│   ├── file_tools.py         # File operations
│   └── code_tools.py         # Code execution
│
├── agents/                    # Multi-agent system
│   ├── __init__.py
│   ├── base.py               # Agent framework
│   ├── code_agent.py         # Code review & refactoring
│   ├── debug_agent.py        # Debugging specialist
│   ├── doc_agent.py          # Documentation generator
│   └── test_agent.py         # Test generation
│
├── plugins/                   # User plugins (auto-created)
│   └── example_plugin.py     # Example plugin template
│
├── memory/                    # Data persistence (auto-created)
│   ├── learning.db           # Learning system database
│   └── chroma/               # Vector database
│
├── sessions/                  # Session data (auto-created)
│   └── history.txt           # Command history
│
├── logs/                      # Application logs (auto-created)
│   └── aria.log              # Main log file
│
├── requirements.txt           # Python dependencies
├── .env.example               # Configuration template
├── .env                       # User configuration (create from .env.example)
├── .gitignore                # Git ignore rules
│
├── README.md                  # Original project specification
├── QUICKSTART.md             # Quick start guide
├── ADVANCED_FEATURES.md      # Advanced features documentation
└── setup.sh                  # Ubuntu setup script
```

🔧 Core Components

Main Application (`aria.py`)

- REPL interface with Rich formatting
- Command parsing and routing
- Integration of all systems
- Streaming response handling

Configuration (`config.py`)

- Pydantic-based settings
- Environment variable loading
- Directory management
- Feature toggles

LLM Interface (`llm_interface.py`)

- Ollama client wrapper
- Streaming chat support
- Conversation history
- Context management
- Model switching
- Embedding generation

🔌 Plugin System (`plugin_system.py`)

Features:

- Dynamic plugin discovery
- Hot-reloading
- Plugin lifecycle hooks (on_load, on_unload)
- Tool registration
- Message interception

Plugin Structure:

```python
class MyPlugin(Plugin):
    metadata = PluginMetadata(...)

    def on_load(self): ...
    def get_tools(self): ...
    def on_message(self, message): ...
```

🤖 Multi-Agent System (`agents/`)

Architecture:

- Base agent framework
- Capability-based routing
- Confidence scoring
- Specialized implementations

Agents:

1. **CodeReviewAgent** - Security, performance, best practices
2. **RefactoringAgent** - Design patterns, SOLID principles
3. **DebugAgent** - Error analysis, stack traces
4. **DocumentationAgent** - Docstrings, READMEs
5. **TestGenerationAgent** - Unit test creation

Agent Coordinator:

- Task routing
- Confidence evaluation
- Agent selection
- Response management

🧠 Learning System (`learning_system.py`)

Components:

- Feedback collection (corrections, ratings)
- Pattern extraction
- User preferences
- SQLite persistence
- Prompt enhancement

Database Schema:

```sql
feedback     - User feedback entries
patterns     - Learned patterns
preferences  - User preferences
```

🛠️ Tool System (`tools/`)

Base Framework:

- Tool abstract base class
- ToolRegistry for management
- ToolResult for outputs

Built-in Tools:

- ReadFileTool - File reading
- WriteFileTool - File writing
- ListFilesTool - Directory listing
- ExecuteCodeTool - Code execution

📊 Data Flow

```plaintext
User Input
    ↓
Command Parser
    ↓
    ├→ Tool Execution
    ├→ Agent Routing
    ├→ Plugin Hooks
    └→ Chat Processing
            ↓
    Learning Enhancement
            ↓
    LLM Processing
            ↓
    Streaming Response
            ↓
    Learning Storage
```

🚀 Key Features

Phase 1: Core REPL ✅

- Terminal interface with Rich
- Streaming responses
- File operations
- Code execution
- Model management

Phase 2: Tools & Execution ✅

- Multi-language code execution
- Safe execution environment
- Output capture

Phase 3: Advanced Features ✅

- **Plugin System**: Extensible architecture
- **Multi-Agent**: Specialized capabilities
- **Learning**: Continuous improvement

🔐 Security Considerations

1. **Code Execution:**
    - Timeout protection
    - Temporary file isolation
    - Can be disabled via config
2. **File Operations:**
    - Path validation
    - Error handling
    - User confirmation for writes
3. **Plugin System:**
    - Sandboxed loading
    - Error isolation
    - Can be disabled

📈 Performance Notes

- **Streaming**: Responses stream token-by-token
- **Context Management**: Auto-pruning to stay within limits
- **Plugin Lazy Loading**: Plugins load only when needed
- **Database**: SQLite for lightweight persistence

🎯 Extension Points

1. **Custom Tools**: Add to `tools/` directory
2. **Custom Agents**: Add to `agents/` directory
3. **Plugins**: Add to `plugins/` directory
4. **Commands**: Extend `_parse_command()` in `aria.py`

🔄 Future Enhancements

Potential additions (mentioned in README):

- Vector-based semantic memory (ChromaDB ready)
- Git integration
- Project analysis
- Voice integration
- Web UI
- IDE plugins
- Multi-agent collaboration
- Performance monitoring

📝 Configuration Options

```env
# Core
OLLAMA_HOST=http://localhost:11434
DEFAULT_MODEL=llama3

# Features
ENABLE_PLUGINS=true
ENABLE_AGENTS=true
ENABLE_LEARNING=true
ENABLE_CODE_EXECUTION=true

# Limits
MAX_CONTEXT_MESSAGES=50
SAFE_MODE=true
```

🧪 Testing Strategy

Manual Testing:

1. Core commands (@read, @write, @exec, @ls)
2. Agent routing (@review, @refactor, @debug, @test, @document)
3. Plugin loading (/plugins)
4. Learning system (/correct, /rate, /learn)
5. Model switching (@model)

Integration Points:

- Plugin tool registration
- Agent capability detection
- Learning prompt enhancement
- Message hook interception

📚 Documentation

- **README.md**: Project overview and goals
- **QUICKSTART.md**: Installation and basic usage
- **ADVANCED_FEATURES.md**: Detailed feature documentation
- **Code Comments**: Inline documentation

🎓 Learning Path

1. Start with basic commands (QUICKSTART.md)
2. Explore file operations and code execution
3. Try specialized agents
4. Create a custom plugin
5. Use learning system for personalization
6. Read ADVANCED_FEATURES.md for deep dive

Built with ❤️ for the coding community