ATLAS - Adaptive Real-time Intelligence Assistant

🚀 Project Overview

ATLAS is a JARVIS-like AI intelligent assistant that lives in your terminal, powered by local Ollama language models. It's designed to be your intelligent companion for software development, offering real-time responses, file operations, code execution, web automation, and advanced multi-agent capabilities—all while maintaining conversation memory and learning from your feedback.

Think of ATLAS as your personal AI developer sitting right in your terminal, ready to help with everything from reading files and executing code to debugging complex issues and conducting web research.

🎯 Core Philosophy

ATLAS is built on three fundamental principles:

1. **Local-First**: Uses Ollama for complete privacy and control—your code never leaves your machine
2. **Adaptive Intelligence**: Learns from your corrections and preferences to provide increasingly personalized assistance
3. **Terminal-Native**: Seamlessly integrates into your development workflow without leaving the command line

---

✨ Key Features

1. Interactive Terminal Interface

- **REPL-Style Shell**: Persistent, responsive conversation interface
- **Real-Time Streaming**: See AI responses generate token-by-token
- **Rich Formatting**: Markdown rendering, syntax highlighting, and colored output
- **Command History**: Navigate previous commands with full history support
- **Multi-line Input**: Handle complex queries and code snippets

2. Ollama Integration

- **Model Flexibility**: Switch between models (llama3, codellama, mistral, etc.) on the fly
- **Streaming Responses**: Live token generation for natural interaction
- **Smart Context Management**: Automatically manages conversation context to avoid token limits
- **Temperature Control**: Adjust AI creativity/determinism as needed

3. File System Operations

- **Read Files**: Parse and understand various file types (code, markdown, JSON, logs)
- **Write Files**: Create, modify, and append to files with user confirmation
- **Directory Listing**: Navigate and understand project structures
- **Format Support**: Intelligent handling of different file formats

4. Code Execution Engine

- **Multi-Language Support**: Execute Python, JavaScript, Bash, and more
- **Real-Time Output**: Stream stdout/stderr as code runs
- **Error Handling**: Catch errors, explain them, and suggest fixes
- **Safe Execution**: Isolated execution environment

5. Web Automation & Search

- **Browser Automation**: Control headless Chromium with Selenium
  - Navigate to URLs
  - Click elements
  - Type into forms
  - Extract text
  - Take screenshots
- **AI-Powered Search**: Tavily integration for intelligent web search with citations
- **Web Scraping**: Extract content from any web page
- **Data Extraction**: Pull structured data (links, images, tables) from websites

6. Multi-Agent System 🤖

ATLAS includes specialized agents that excel at specific tasks:

- **CodeReviewAgent**: Expert code review focusing on:

  - Security vulnerabilities
  - Performance optimization
  - Best practices
  - Architecture patterns

- **RefactoringAgent**: Refactoring specialist providing:

  - Design pattern recommendations

  - SOLID principle application
  - Code smell detection
  - Maintainability improvements

- **DebugAgent**: Debugging expert offering:

  - Stack trace analysis
  - Error interpretation
  - Root cause identification
  - Fix suggestions

- **DocumentationAgent**: Documentation specialist for:

  - Docstrings
  - README files
  - API documentation
  - Code comments

- **TestGenerationAgent**: Test creation expert providing:

  - Unit test generation
  - Edge case identification
  - Mocking strategies
  - Coverage optimization

7. Plugin System 🔌

Extend ATLAS with custom functionality:

- **Dynamic Loading**: Plugins load automatically from the `plugins/` directory
- **Hot Reloading**: Reload plugins without restarting ATLAS
- **Custom Tools**: Add domain-specific tools and commands
- **Message Hooks**: Intercept and process user messages
- **API Integrations**: Connect to external services

8. Learning System 🧠

ATLAS learns from your feedback:

- **Feedback Collection**: Records corrections and ratings in SQLite
- **Pattern Recognition**: Identifies similar corrections and stores patterns
- **Adaptive Responses**: Enhances future responses based on learned preferences
- **User Preferences**: Remembers coding style, frameworks, and patterns
- **Statistics Tracking**: Monitor learning progress over time

9. Conversation Memory

- **Session Persistence**: Maintains conversation history
- **Context Awareness**: Remembers previous interactions
- **Smart Pruning**: Removes less relevant messages when approaching token limits
- **Long-Term Memory**: Stores important information across sessions

---

🛠️ Technical Architecture

```plaintext
┌─────────────────────────────────────────┐
│       Terminal Interface (Rich/TUI)     │
│   Real-time streaming • Markdown        │
└─────────────────┬───────────────────────┘
                  │
┌─────────────────▼───────────────────────┐
│       Conversation Manager (ATLAS)       │
│  • Command parsing & routing            │
│  • Plugin message hooks                 │
│  • Session management                   │
└─────────────────┬───────────────────────┘
                  │
        ┌─────────┴─────────┐
        │                   │
┌───────▼────────┐  ┌──────▼──────┐
│  LLM Interface │  │   Learning  │
│   (Ollama)     │  │   System    │
│  • Streaming   │  │  • SQLite   │
│  • Models      │  │  • Patterns │
└───────┬────────┘  └──────┬──────┘
        │                   │
┌───────▼───────────────────▼──────┐
│       Multi-Agent Coordinator     │
│  • Agent routing                  │
│  • Capability matching            │
│  • Confidence scoring             │
└───────┬───────────────────────────┘
        │
┌───────▼───────────────────────────┐
│         Tool Registry             │
│  • File operations                │
│  • Code execution                 │
│  • Web automation                 │
│  • Plugin tools                   │
└───────────────────────────────────┘
```

📋 Command Reference

File Operations

bash

```bash
@read <file>                # Read and display file contents
@write <file> <content>     # Write content to file
@ls [directory]             # List files in directory (default: current)
```

Code Execution

bash

```bash
@exec <code>                # Execute Python code
```

Web Tools

bash

```bash
# Browser Automation
@web navigate <url>         # Navigate to URL
@web click <selector>       # Click element (CSS selector)
@web type <selector> <text>  # Type text into element
@web get_text <selector>    # Get element text content
@web screenshot [file]      # Take screenshot
@web close                 # Close browser session

# Web Search & Scraping
@search <query>            # AI-powered search with Tavily
@scrape <url>              # Scrape web page content
@extract <url> [type]      # Extract data (text/links/images/tables)
```

Specialized Agents

bash

```bash
@review                    # Route to CodeReviewAgent
@refactor                 # Route to RefactoringAgent
@debug                    # Route to DebugAgent
@test                     # Route to TestGenerationAgent
@document                 # Route to DocumentationAgent
```

Model Control

```bash
@model <name>            # Switch Ollama model (llama3, codellama, mistral, etc.)
```

Learning System

```bash
/correct <text>          # Provide correction for last response
/rate <1-5>             # Rate last response (1-5 stars)
/learn                  # Show learning statistics
```

Plugin Management

```bash
/plugins                # List all loaded plugins
/plugin reload <name>   # Hot-reload a specific plugin
```

Session Management

```bash
/memory                 # Show conversation context and memory status
/clear                  # Clear conversation history
/agents                 # List available specialized agents
/help                   # Show help and system status
/exit                   # Exit ATLAS
```

Natural Language

Simply type your questions or requests naturally—ATLAS understands context!

🎬 Example Use Cases

1. Code Review Workflow

```plaintext
You: @read src/api/auth.py

ATLAS:  [Shows file contents]

You: @review this code for security issues

 CodeReviewAgent (confidence: 92%)

## Security Review

### Critical Issues Found:
1. SQL Injection vulnerability on line 45...
2. Hardcoded credentials on line 78...
[detailed analysis with recommendations]

You: /rate 5

ATLAS:  Thank you for rating my response:
```

2. Web Research + Code Generation

```plaintext
You: @search best practices for Python async error handling

ATLAS:  [AI-generated answer with citations]

You: Based on that, can you write an async retry decorator?

ATLAS: [Generates code with detailed explanation]

You: /correct Always include type hints and docstrings

ATLAS:  I've recorded your correction and will learn from it.
```

3. Automated Testing Workflow

```plaintext
You: @web navigate https://myapp.com
You: @web type #username testuser
You: @web type #password testpass
You: @web click button#login
You: @web screenshot logged_in.png
You: @web close

ATLAS:  Screenshot saved to logged_in.png
      Browser closed successfully
```

4. Test Generation

```plaintext
You: @read src/utils/validators.py

You: @test generate unit tests for the email_validator function

 TestGenerationAgent (confidence: 88%)

[Generates comprehensive test suite with edge cases]
```

5. Debugging Session

```plaintext
You: I'm getting a KeyError in my Flask app when processing JSON

ATLAS: Can you share the error traceback?

You: [pastes traceback]

You: @debug

 DebugAgent (confidence: 90%)

## Root Cause Analysis
The error occurs because...
[detailed debugging steps and solution]
```

🏗 Project Structure

```plaintext
ATLAS/
├── ATLAS.py                  # Main application entry point
├── config.py               # Configuration management
├── llm_interface.py        # Ollama integration
├── plugin_system.py        # Plugin management
├── learning_system.py      # Learning and feedback system
├── demo.py                 # Demo/testing script
├── requirements.txt        # Python dependencies
├── setup.sh               # Setup script
├── .env.example           # Environment template
│
├── tools/                 # Tool implementations
│   ├── __init__.py
│   ├── base.py           # Base tool classes
│   ├── file_tools.py     # File operations
│   ├── code_tools.py     # Code execution
│   └── web_tools.py      # Web automation & search
│
├── agents/               # Specialized agents
│   ├── __init__.py
│   ├── base.py          # Base agent classes
│   ├── code_agent.py    # Code review & refactoring
│   ├── debug_agent.py   # Debugging specialist
│   ├── doc_agent.py     # Documentation generator
│   └── test_agent.py    # Test generation
│
├── plugins/             # User plugins directory
│   └── [custom plugins]
│
├── memory/              # Learning database
│   └── learning.db
│
├── logs/                # Application logs
│   └── ATLAS.log
│
└── sessions/            # Session data
    └── history.txt
```

🚦 Quick Start

Prerequisites

- Python 3.8+

- Ollama installed and running
- Ubuntu/Linux (for web tools: Chromium)

Installation

```bash
# 1. Install Ollama
curl -fsSL https://ollama.com/install.sh | sh

# 2. Pull a model
ollama pull llama3

# 3. Setup ATLAS
cd ATLAS
chmod +x setup.sh
./setup.sh

# 4. (Optional) Install web tools
sudo apt install chromium-browser chromium-chromedriver
pip install selenium tavily-python

# 5. Configure (Optional)
cp .env.example .env
nano .env  # Add TAVILY_API_KEY if using web search
```

Running ATLAS

```bash
# Activate virtual environment
source venv/bin/activate

# Run ATLAS
python ATLAS.py
```

🔧 Configuration

Edit `.env` to customize ATLAS:

```env
# Ollama settings
OLLAMA_MODEL=llama3
OLLAMA_BASE_URL=http://localhost:11434

# Feature toggles
ENABLE_PLUGINS=true
ENABLE_AGENTS=true
ENABLE_LEARNING=true

# Web tools
TAVILY_API_KEY=your_api_key_here

# Logging
LOG_LEVEL=INFO
```

🎨 Advanced Features

Creating Custom Plugins

```python
# plugins/my_plugin.py
from plugin_system import Plugin, PluginMetadata
from tools.base import Tool, ToolResult

class MyCustomTool(Tool):
    """Custom tool implementation"""

    def execute(self, **kwargs) -> ToolResult:
        # Your logic here
        return ToolResult(success=True, output="Result")

class MyPlugin(Plugin):
    metadata = PluginMetadata(
        name="My Plugin",
        version="1.0.0",
        author="Your Name",
        description="What it does"
    )

    def get_tools(self):
        return [MyCustomTool()]

    def on_message(self, message: str):
        # Hook into messages
        return None

def load_plugin():
    return MyPlugin()
```

Creating Custom Agents

python

```python
from agents.base import Agent, AgentTask, AgentResponse

class MyCustomAgent(Agent):
    def __init__(self, llm_interface):
        super().__init__("MyAgent", llm_interface)
        self.system_prompt = "You are a specialist in..."

    def can_handle(self, task: AgentTask) -> float:
        if "keyword" in task.content.lower():
            return 0.9  # High confidence
        return 0.0

    def process(self, task: AgentTask) -> AgentResponse:
        response = self.llm.chat(task.content, self.system_prompt)
        return AgentResponse(
            success=True,
            content=response,
            confidence=0.9,
            agent_name=self.name
        )
```

📊 Technical Stack

Core:

- Python 3.8+
- Ollama (Local LLM runtime)

Libraries:

- `rich` - Beautiful terminal UI
- `prompt_toolkit` - Advanced input handling
- `ollama` - Ollama API client
- `selenium` - Web automation
- `tavily-python` - AI web search
- `sqlalchemy` - Learning database ORM

🎯 Implementation Status

✅ Completed Features

Phase 1-3: Core Functionality

- ✅ Interactive REPL with real-time streaming
- ✅ File operations (read/write/list)
- ✅ Code execution (Python, JavaScript, Bash)
- ✅ Ollama integration with model switching
- ✅ Rich terminal formatting
- ✅ Conversation history management

Phase 4-5: Advanced Features

- ✅ Multi-agent system with 5 specialized agents
- ✅ Plugin system with hot-reloading
- ✅ Learning system with feedback collection
- ✅ Pattern recognition and preference storage
- ✅ SQLite-based memory persistence

Phase 6: Web Tools

- ✅ Browser automation (Selenium + headless Chromium)
- ✅ AI-powered web search (Tavily)
- ✅ Web scraping and content extraction
- ✅ Structured data extraction (links, images, tables)

Bonus Challenges Completed:

- ✅ Plugin system
- ✅ Multi-agent system
- ✅ Learning mode

🔮 Future Enhancements

Planned features:

- [ ] Voice integration (speech-to-text, text-to-speech)
- [ ] Web UI companion interface
- [ ] IDE integrations (VSCode, Neovim)
- [ ] Vector embeddings for better pattern matching
- [ ] Agent collaboration on complex tasks
- [ ] Plugin marketplace/discovery
- [ ] Export/import learned patterns
- [ ] Visual learning dashboard
- [ ] Project scaffolding templates
- [ ] Git integration (commits, diffs, analysis)

🐛 Troubleshooting

Connection refused error:

```bash
# Check Ollama status
systemctl status ollama

# Start Ollama if needed
ollama serve
```

Model not found:

```bash
# Pull the model
ollama pull llama3

# List available models
ollama list
```

Web tools not working:

```bash
# Install Chromium (Ubuntu/Debian)
sudo apt install chromium-browser chromium-chromedriver

# Verify installation
which chromium-browser
which chromedriver
```

Import errors:

```bash
# Ensure virtual environment is active
source venv/bin/activate

# Reinstall dependencies
pip install -r requirements.txt
```

📖 Documentation

- README.md: Project overview and architecture
- QUICKSTART.md: Quick start guide
- ADVANCED_FEATURES.md: Plugin system, agents, and learning
- WEB_TOOLS.md: Web automation and search documentation
- PROJECT_STRUCTURE.md: Detailed project structure

💡 Best Practices

General Usage

- Start with `/help` to see all available commands
- Use `@read` before asking questions about code
- Provide context for better responses
- Use specialized agents (`@review` , `@debug` , etc.) for specific tasks

Learning System

- Provide specific corrections, not generic feedback
- Rate responses to help ATLAS improve
- Use corrections for preferences, not one-off fixes
- Check `/learn` periodically to see progress

Web Tools

- Always close browsers with `@web close` when done
- Respect robots.txt and website ToS
- Add delays between requests for rate limiting
- Use descriptive user agents

Plugins

- Keep plugins focused on one task
- Use descriptive names and document your code
- Handle errors gracefully
- Test plugins before deploying

🤝 Contributing

ATLAS is designed to be extensible. Contributions welcome:

- Create custom plugins
- Develop new specialized agents
- Improve existing tools
- Add documentation
- Report bugs and suggestions

📜 License

[Add your license information here]

🙏 Acknowledgments

Built with:

- **Ollama** - Local LLM runtime
- **Rich** - Terminal formatting
- **Selenium** - Web automation
- **Tavily** - AI web search

📞 Support

For issues, questions, or feature requests, please check:

- Documentation in the `/docs` directory
- Logs in `logs/ATLAS.log`
- Use `/help` within ATLAS for command reference

ATLAS: Your intelligent terminal companion for modern software development. 🚀🤖

Built with ❤️ for developers who live in the terminal.