Local Python Agent Prompt: AI Coder Orchestration System

Task Overview

Build a Python system that acts as an intelligent orchestrator for AI coding assistants (like Cursor AI). This system should manage project context, break down tasks, provide structured feedback, and keep AI coders on track.

Core Requirements

1. Project State Management

- Track all project files and their current state
- Maintain a clear hierarchy of goals and sub-goals
- Persist state between sessions
- Monitor file system changes in real-time

2. AI Interface Layer

- Create adapters for different AI coding tools (Cursor, GitHub Copilot, etc.)
- Format prompts with proper context and constraints
- Parse and validate AI responses
- Handle API rate limits and errors gracefully

3. Quality Assurance System

- Validate file references (prevent hallucination)
- Check code syntax and basic quality
- Run automated tests when available
- Detect when AI gets stuck in loops

4. Feedback Generation

- Generate corrective prompts for common issues
- Provide alternative approaches when stuck
- Break down complex tasks into smaller steps
- Maintain focus on current objectives

Specific Implementation Details

File Structure

```
ai_orchestrator/
├── core/
│   ├── orchestrator.py     # Main orchestration logic
│   ├── state_manager.py    # Project state tracking
│   ├── file_monitor.py     # File system monitoring
│   └── goal_manager.py     # Goal decomposition
├── ai_interfaces/
│   ├── cursor_interface.py # Cursor AI integration
│   ├── base_interface.py   # Abstract base class
│   └── mock_interface.py   # Testing interface
├── quality/
│   ├── validator.py        # Code and reference validation
│   ├── test_runner.py      # Automated testing
│   └── syntax_checker.py   # Language-specific checks
└── utils/
    ├── prompt_builder.py   # Structured prompt generation
    └── feedback_generator.py # Corrective feedback
```

Key Features to Implement

1. **Smart Context Management**
   - Only include relevant files in prompts (avoid context overflow)
   - Summarize large files or provide targeted excerpts
   - Track dependencies between files
2. **Goal Decomposition Engine**
   - Automatically break large goals into sub-goals
   - Estimate complexity and time requirements
   - Reorder tasks based on dependencies
3. **Progress Tracking**
   - Measure concrete progress (lines of code, tests passing, features complete)
   - Identify when progress stalls
   - Generate progress reports
4. **Error Recovery**
   - Detect common failure patterns
   - Automatically retry with modified prompts

  - Escalate to human when stuck

Integration Points

Cursor AI Integration

- Use Cursor's API if available
- Monitor Cursor's workspace for changes
- Send structured prompts through appropriate channels
- Parse Cursor's responses for code and explanations

Development Environment

- Integrate with git for version control
- Run linters and formatters automatically
- Execute tests and capture results
- Monitor build processes

File System Monitoring

- Watch for file changes in real-time
- Detect new files, deletions, and modifications
- Trigger re-evaluation when project state changes

Example Usage Scenario

python

```python
# Initialize the orchestrator
orchestrator = AIOrchestrator(
    project_path="/path/to/my/project",
    ai_tool="cursor",
    project_goal="Build a REST API for task management"
)

# Start the orchestration process
orchestrator.start_session()

# The system will:
# 1. Analyze current project state
# 2. Break down the goal into sub-goals
# 3. Generate appropriate prompts for Cursor
# 4. Monitor Cursor's responses and file changes
# 5. Provide feedback and corrections as needed
# 6. Track progress toward the overall goal
```

Technical Considerations

Performance

- Efficient file monitoring (use OS-level watchers)
- Smart context window management
- Asynchronous processing where possible

Reliability

- Robust error handling and recovery
- State persistence across crashes
- Graceful degradation when AI services are unavailable

Extensibility

- Plugin architecture for different AI tools
- Configurable quality checks and validators
- Customizable prompt templates

Success Metrics

The system should be able to:

- Keep AI coders focused on specific sub-goals
- Prevent common issues like file hallucination
- Automatically recover from typical failure modes
- Provide meaningful progress tracking
- Reduce the need for manual intervention

Implementation Priority

1. **Phase 1**: Basic orchestration with mock AI interface
2. **Phase 2**: Real AI integration (starting with Cursor)
3. **Phase 3**: Advanced features (goal decomposition, progress tracking)
4. **Phase 4**: Multi-AI support and advanced quality checks

Please implement this system with a focus on modularity and testability. Start with the core orchestration logic and build out the features incrementally.