AI Sidecar Command Agent

Experimental MVP – Product Specification & Commercial Prospectus  
Version 1.2 – August 2025

# 1. Executive Summary

This proposal introduces the AI Sidecar Command Agent (AISA), a lightweight local execution framework that extends the utility of AI tools such as ChatGPT. AISA interprets structured command blocks issued by AI and executes them in a secure, sandboxed local environment to manage files, track lineage, and perform Git operations.  
  
While designed as an experimental proof-of-concept MVP, AISA has profound long-term potential for:  
- Enabling AI-assisted version control and publishing  
- Reducing human bottlenecks in coding, writing, and content workflows  
- Introducing traceability and reversibility to AI outputs  
- Forming the basis for compliant, auditable AI-human collaboration systems  
- Seeding future cooperative AI protocols where human intent is structured and validated before execution

# 2. Introduction

Modern AI tools are powerful but stateless—they lack persistent memory, file access, and context awareness across sessions. This tool acts as a local 'sidecar brain' for AI, interpreting structured instructions ([COMMAND] blocks) and handling local file operations, Git interactions, and metadata tracking in a traceable, controllable manner.

# 3. Functional Requirements

|  |  |  |
| --- | --- | --- |
| Requirement | Status | Description |
| Parse [COMMAND] blocks | MVP | Extract structured directives embedded in plain text or .md files. |
| Execute file and Git ops | MVP | Run file operations (create/move/edit/delete) and Git commits. |
| Track lineage/logs | MVP | Maintain changelogs for each operation set. |
| Sandboxed directories | MVP | Ensure all actions remain within a known local structure. |
| Dry-run & rollback | MVP | Simulate commands before execution; enable undo logs. |
| Manual override | MVP | Allow users to approve or modify generated actions. |

# 4. Non-Functional Requirements

- Cross-platform support (Linux, macOS, Windows)  
- Python 3.8+ based; minimal third-party dependencies  
- Configurable safety settings (no external access)  
- Optional telemetry for usage feedback (opt-in)  
- Expandable plugin-based architecture  
- Future GUI/visual interface

# 5. System Architecture

[Diagram Placeholder: Modular architecture diagram showing Command Parser, Shell/Git Executor, Audit Logger, Metadata Tracker (JSON), optional Feedback Agent]

# 6. Build Strategy and Work Estimates

Implementation will be divided into the following phases, using modular testable components:  
- CLI Parser & Sandbox Setup: 6–8 hours  
- Git Command Wrapper (Commit, Pull, Push): 5–7 hours  
- Logging Engine & Dry-run Preview: 5–6 hours  
- JSON Metadata Tracker: 3–4 hours  
- Command Validation & Manual Approval Interface: 4–5 hours  
- Optional GUI (Tkinter or Electron shell): 20–40 hours  
- Feedback Agent or Plugin Protocols: 10–12 hours  
  
Total MVP Timeline: ~25–30 hours (~1–2 developers over a week)

[Remaining sections will be regenerated from prior content and included below.]

# Table of Contents

[TOC will be generated in Word]

# 7. Setup Instructions

Install Git and Python 3.8+.  
Create folders: /incoming, /staging, /archive, /registry.  
Run script to parse and execute [COMMAND]s.  
Review logs after batch.

# 8. Commercial Viability

Target Users:  
- AI creators  
- Substack writers  
- Git maintainers  
- Civic tech contributors

Revenue Options:  
- Free CLI version  
- Paid Pro with GUI & plugin API  
- Tiered licensing: $200–500/user

Projected Phase 1 Revenue:  
- 500 users @ $200 = $100,000  
- Add-ons: LTV ↑  
- Enterprise site licenses: $2k+/client

# 9. Marketing Strategy

Channels:  
- Hacker News  
- Product Hunt  
- Reddit (r/ChatGPT, r/LocalLLaMA, r/codingtools)

Tactics:  
- Screencasts of AI controlling local repos  
- SEO: 'Give ChatGPT memory', 'AI Git assistant'  
- Community posts for early adopters  
- Substack/Discord beta access

# 10. Branding & Naming

Possible Names:  
- AISA (AI Sidecar Agent)  
- GhostHand  
- PromptPilot  
- CmdBridge  
- EchoDO

UX Style: retro-terminal + clean overlays  
[Insert logo concept here]

# 11. UI Mockups (Planned)

[Insert mockup sketches or UI shots]  
- Command Queue Panel  
- File History Tree Viewer  
- Session Playback Dashboard

# 12. Financial Forecast

Phase 1 Targets:  
- Open Source: Adoption only  
- MVP: 200 users x $200 = $40k  
- Pro Add-ons: 50 x $500 = $25k  
- Enterprise: 5 x $2k = $10k  
Total Phase 1: $75k+

# 13. Competitive Landscape

- Notion AI: No file or Git ops  
- LangChain: Dev-centric, complex setup  
- Copilot: IDE-bound, no workflow autonomy

AISA offers: Lightweight, structured, local execution

# 14. Future Expansion

- GPT plugin integration  
- Mobile/voice control agents  
- Use-case kits: Substack, repo curation, publishing pipelines  
- Auto-archival by AI-driven usage

# 15. Call to Action

Looking for collaborators:  
- Python or Node dev  
- UX/UI designer  
- Early users for feedback and promo  
Email: rick@groupbuild.org

## 5.1 Architecture Diagram

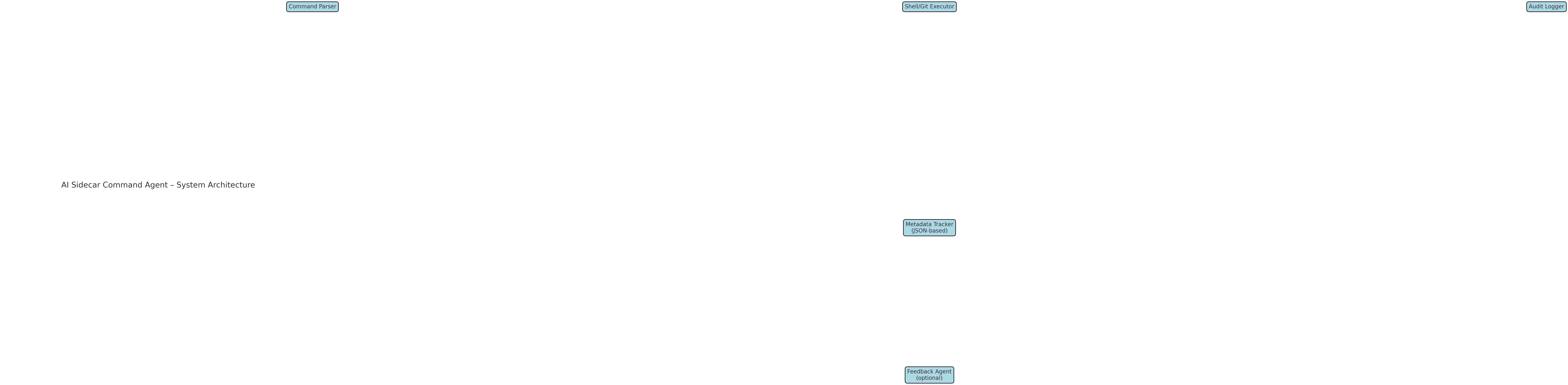


Figure: High-level overview of AISA component relationships.

# 16. Developer Onboarding Instructions

1. 1. Clone the private GitHub repository (or work locally as instructed) and set up a Python 3.8+ environment.
2. 2. Create and verify the local folder structure: /incoming, /staging, /archive, /registry.
3. 3. Implement a [COMMAND] parser to extract instructions from Markdown (.md) files.
4. 4. Write secure Python functions to perform basic file operations: create, move, delete, edit.
5. 5. Wrap Git commands in Python using subprocess: commit, push, pull. Ensure proper logging.
6. 6. Implement a dry-run mode to simulate actions, and build a rollback log (e.g., JSON-based).
7. 7. Track every operation with a metadata logger. Create a human-readable log and a machine-readable state.
8. 8. Validate [COMMAND] inputs before execution. Include a manual override system (Y/n prompt).
9. 9. Optional: begin laying the structure for a GUI (Tkinter or Electron shell). Keep interface modular.
10. 10. Use minimal third-party libraries. Aim for high transparency, portability, and safety.
11. 11. Document all components clearly. Use inline docstrings and a separate README.md with setup steps.
12. 12. Submit work for approval in stages. Each phase should be testable and revertible.

Note: Developer must sign NDA and IP assignment agreement before any code is deployed.