Dual-Mode Agentic System Architecture: Technical Design & Implementation Report

Executive Summary

The convergence of local Large Language Models (LLMs), highly optimized hardware such as Apple Silicon, and sophisticated agentic frameworks has enabled the creation of personalized **Agentic Operating Systems (AOS)**. This report outlines the comprehensive technical design for a dual-mode local agent system tailored to specific user requirements: a "Desktop Organizer" (Mode A) designed for ADHD executive function support, and a "Trading Agent" (Mode B) designed for algorithmic trading with strict risk management. The system is architected to run locally on a MacBook Air, leveraging local inference via Ollama to ensure privacy, low latency, and operational security.

The core design philosophy treats the system not merely as a chatbot, but as a proactive system overlay. It employs a "Dispatcher" pattern to strictly isolate the two modes, ensuring that the permissive file-system access required by the Organizer never bleeds into the financial decision-making logic of the Trader, and conversely, that the high-risk financial tools are never accessible during routine desktop maintenance. This separation is critical for security and cognitive clarity.

For Mode A, the system utilizes heuristic-based file classification, "safe-by-default" transactional file operations with rollback capabilities, and a "nosey," gamified persona powered by long-term memory (Mem0) to combat ADHD-related executive dysfunction. It acts as a "body double," providing external motivation and reducing the cognitive load of organization. It is context-aware, incorporating the user's personal life—specifically the presence of three children and the startup venture "Tobie.team"—to schedule tasks appropriately.

For Mode B, the system implements a robust algorithmic trading framework designed to transition from paper trading Bitcoin (crypto) to live trading SPY options. The architecture emphasizes a "Risk-First" approach, integrating hard-coded circuit breakers, daily loss limits, and a human-in-the-loop (HITL) approval workflow for all execution logic during the validation phases. The data layer leverages legitimate, compliant APIs (Tradier, Alpaca) to avoid the legal and stability pitfalls of web scraping, specifically addressing the recent T+1 settlement changes in US markets.

This document serves as a blueprint for implementation, providing architectural diagrams, Python code skeletons, configuration schemas, and operational protocols.

Top Recommendations & Tech Stack

Before delving into the architectural minutiae, the following core technologies and providers are recommended as the "Golden Path" for this implementation. These selections prioritize local execution, Python ecosystem compatibility, and regulatory compliance.

Recommended Stack

Component	Recommendation	Justification
Agent Framework	**LangGraph**	Provides superior control over state loops (cyclic graphs) compared to linear chains, essential for the "Trade -> Monitor -> Update" loop and complex decision trees.[1]
Local Inference	**Ollama**	The most efficient runner for Apple Silicon (Metal acceleration). We recommend llama3 or mistral-nemo for the balance of reasoning capability and speed.[3]
Memory Layer	**Mem0**	A dedicated memory management layer that handles vector storage (Chroma/Qdrant) and retrieval, crucial for maintaining the "User Graph" (kids, startup, schedule).[4]
Voice Interface	**Faster-Whisper**	A CTranslate2 implementation of OpenAI's Whisper. It runs locally on CPU/GPU and is roughly 4x faster than the standard

		implementation, enabling near real-time command processing.[6]
Broker (Crypto)	**Alpaca Markets**	Offers a robust, free Paper Trading environment and compliant API for crypto. Excellent Python SDK.[7]
Broker (Options)	**Tradier**	Provides free API access for brokerage account holders. Supports complex multi-leg option orders (vital for SPY spreads) and real-time data streaming.[8]
Safety Layer	**SQLite + Hash**	A local relational database used solely for transaction journaling (undo logs) and audit trails, ensuring every file move or trade is reversible or traceable.

Section 1: System Architecture

1.1 The "Agentic Operating System" Concept

The proposed system is not a standalone script but a persistent service—a daemon—that acts as an intelligent intermediary between the user and the raw capabilities of the OS and the market. The architecture must solve the "Context Pollution" problem: ensuring that the chaotic, creative, and permissive nature of file organization does not introduce instability into the disciplined, high-risk environment of algorithmic trading.

To achieve this, the system follows a **Hub-and-Spoke** architecture, where a central "Dispatcher" node acts as the traffic controller, routing user intents (voice or text) to isolated "Worker" agents. This design ensures the Principle of Least Privilege (PoLP), as each worker agent only possesses the tools and permissions necessary for its specific domain.

1.2 The Dispatcher Pattern

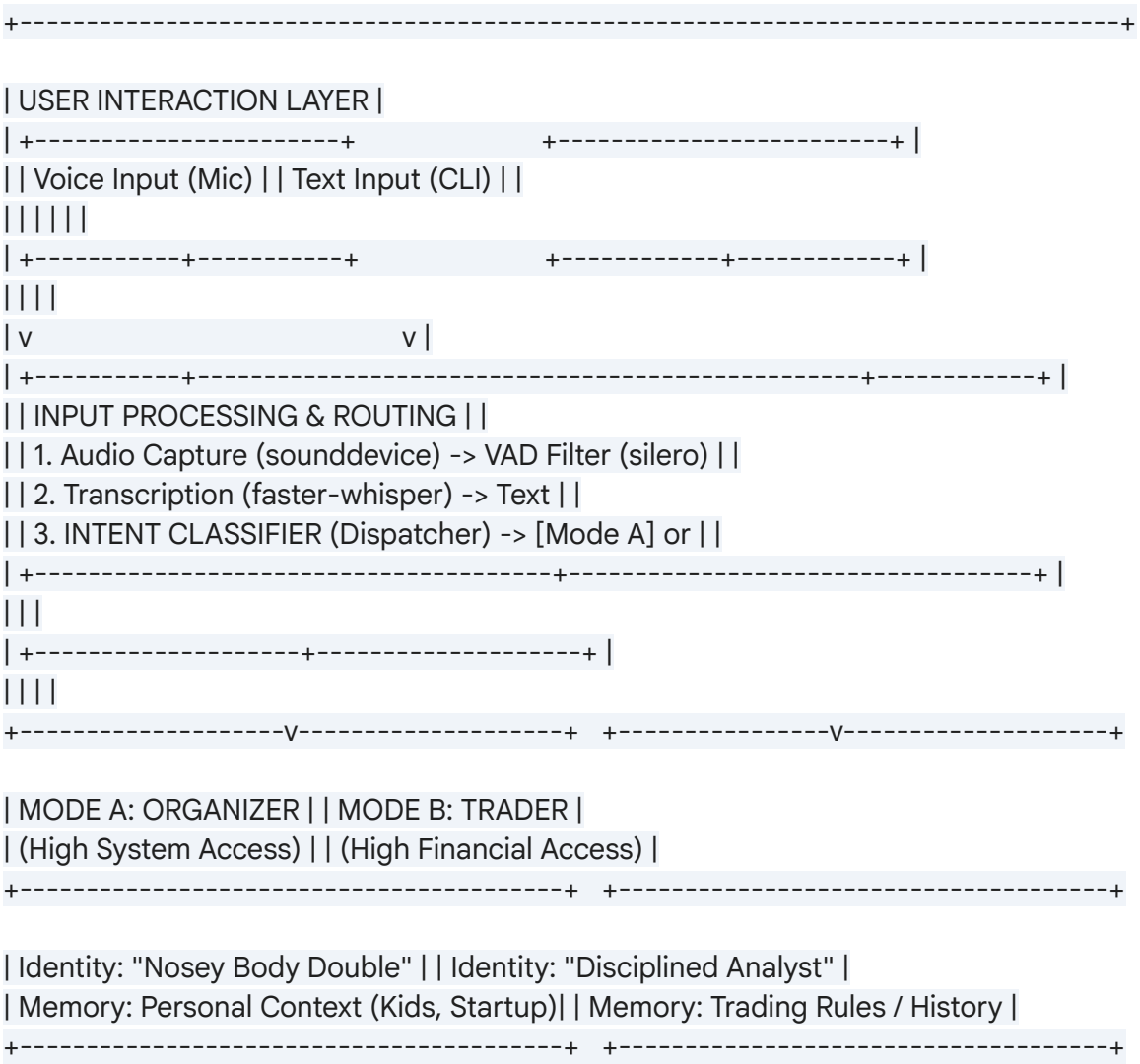
The Dispatcher is a lightweight classifier agent. Its sole responsibility is to analyze the user's input and determine the active context. It maintains the "Mode State" (A vs. B) and gates

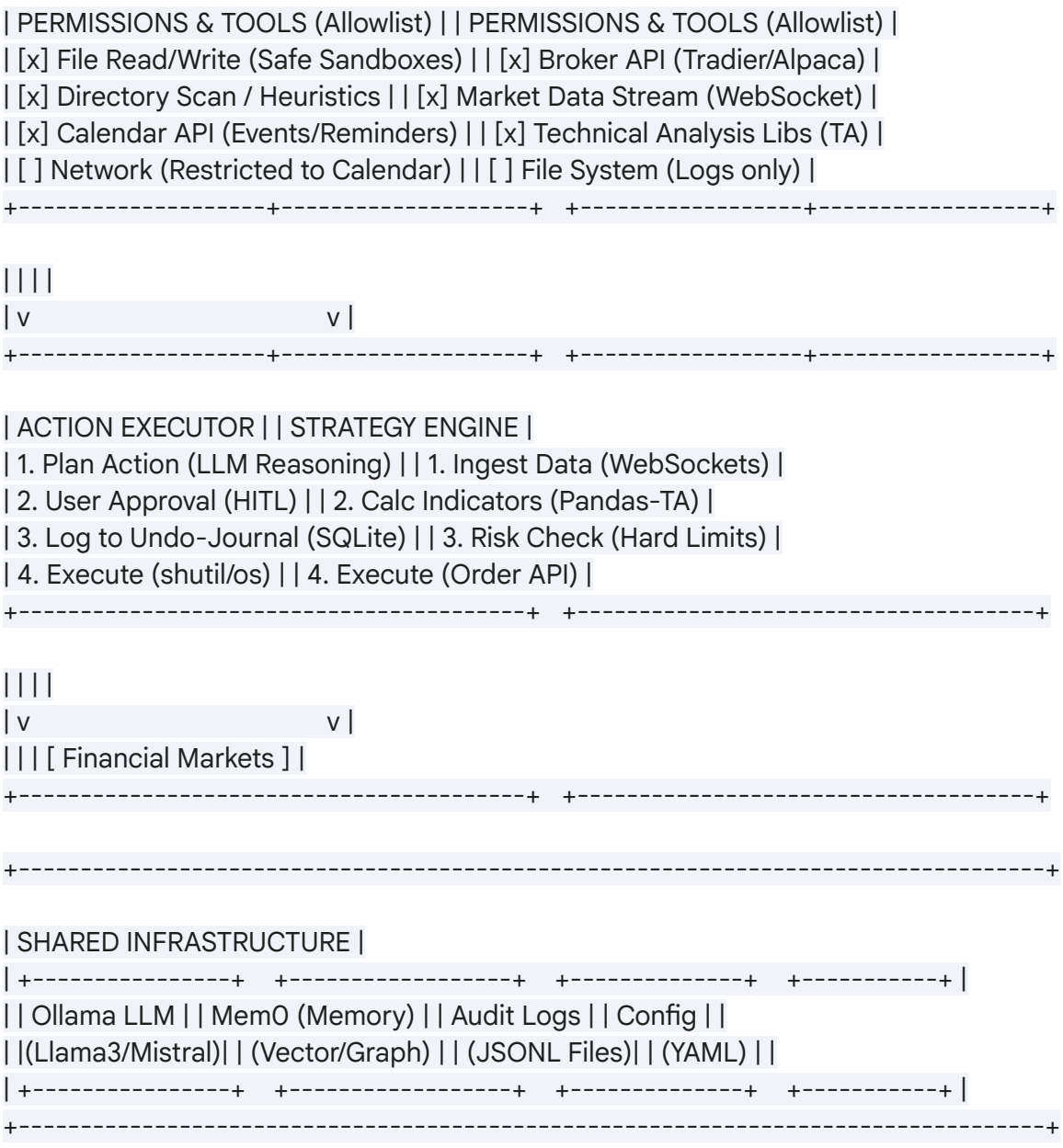
access to the downstream agents.[3]

- **Input**: User voice command (transcribed) or text input.
- **Logic**: Intent classification (e.g., "Is this a file operation, a calendar update, or a market inquiry?"). It utilizes keyword matching and semantic routing.
- **Output**: Routes the context to the OrganizerAgent or TradingAgent.
- **Security**: The Dispatcher *cannot* execute tools. It can only invoke agents. It serves as the primary firewall against context leakage.

1.3 High-Level Architecture Diagram

Code snippet





1.4 Shared Services Breakdown

1.4.1 Local Inference Engine (Ollama)

The "brain" of the system is a local LLM running via Ollama.

- **Model Selection**: llama3:8b or mistral-nemo. These models offer a high balance of reasoning capability and inference speed on Apple Silicon. The 8GB RAM of a base M1/M2 Air is sufficient for 8B parameter models, though 16GB is preferred for multitasking.
- **Context Management**: The system maintains separate "System Prompts" for each

mode. When the Dispatcher routes to Mode A, the context window is injected with the "Organizer" persona. When routing to Mode B, it is cleared and replaced with the "Trader" persona to prevent hallucination overlap.

1.4.2 State & Memory Management (Mem0)

To support the "nosey" and "personal" aspects of the ADHD organizer, the system requires persistent, evolving memory.

- **Mem0 Integration**: Mem0 is utilized to store user preferences and facts.[4]
- **Fact Extraction**: The system automatically extracts entities from conversations. For example, if you mention "Tobie.team needs a logo," Mem0 stores Entity: Tobie.team, Relation: User's Company, Status: Active. This allows the agent to later suggest folders like TobieTeam_Marketing instead of generic names.
- **Personal Context**: Facts like "Has 3 kids" and "Stay-at-home dad schedule" are stored here. This informs the agent's timing—e.g., suggesting cleanups during known "quiet hours" or avoiding trading notifications during school pickup times.[11]

1.4.3 Storage Layer

- **Relational DB (SQLite)**: Used for the "Transaction Journal." Every file move, rename, or trade execution is logged here *before* execution to enable atomic rollbacks and audit trails.[12] This is the "Undo" safety net.
- **Vector Store (ChromaDB or Qdrant via Mem0)**: Stores semantic memories and RAG (Retrieval-Augmented Generation) documents. This is where the trading books and research papers will be indexed for the Trading Agent to reference.
- **Audit Logs (JSONL)**: Immutable, append-only logs of every action taken by the system, designed for human review and debugging.

Section 2: Mode A (Desktop Organizer) Detailed Design

2.1 Design Philosophy: The "Body Double"

For a user with ADHD, the primary friction points are **decision fatigue** and **initiation paralysis**. Mode A is designed not just to organize files, but to provide the *executive function* scaffolding required to start and finish tasks.

- **Persona**: "Nosey," friendly, persistent, and slightly playful. It mimics a "body double"—a person who sits with you while you work to keep you on track.[13]
- **Interaction Model**: Proactive. It doesn't just wait for commands; it "sweeps" the desktop daily and prompts the user: "I found 15 screenshots from yesterday. Want to tackle them in a 60-second speed run?"

2.2 Functional Workflow

2.2.1 Phase 1: Discovery & Inventory

The agent scans targeted directories (Desktop, Downloads) to build an inventory of "clutter." It performs this scan rapidly using Python's os.scandir for performance.

- **Heuristics**:
  - **Stale Files**: Accessed > 30 days ago.
  - **Screenshot Piles**: Regex matching default OS screenshot naming patterns (e.g., Screenshot YYYY-MM-DD...).
  - **Downloads Dump**: Installers (.dmg, .pkg) and temporary zips.
  - **Orphaned Projects**: Folders with low file counts or no recent activity.
  - **Contextual Clusters**: Uses semantic analysis (via LLM) of filenames to group items. E.g., budget.xls, invoice_jan.pdf, and receipt.jpg are clustered as "Financials".

2.2.2 Phase 2: Recommendation Engine

Using the inventory, the LLM proposes a taxonomy.

- **Contextual Naming**: Instead of generic "Folder 1," it uses content analysis to propose Client_TobieTeam_Q1_Assets.
- **Micro-Batching**: To avoid ADHD overwhelm, it presents tasks in batches of 5-10 items max. "Let's just sort these 5 PDFs. Yes/No?" This leverages the "chunking" strategy effective for neurodivergent minds.[13]

2.2.3 Phase 3: Safe Execution with Rollback

File operations are destructive. Safety is paramount.

- **Transaction Logic**:
  1. **Record**: Log source path, intended destination, timestamp, and SHA-256 hash to SQLite.
  2. **Verify**: Check if destination exists; handle collision (auto-rename file_v2).
  3. **Execute**: Perform shutil.move.[15]
  4. **Confirm**: Update log status to "Success".
- **The "Undo" Button**: A global "Oops" command triggers the RollbackService, which reads the last N transactions from SQLite and reverses the moves.[16]

2.3 ADHD-First UX Features

- **Gamification**: "You cleared 1.2GB of clutter today! That's a 'Clean Desktop' streak of 3 days." This targets the dopamine reward system.
- **Nosey Prompts**: "Hey, I see you're on camera. I'll pause notifications so you can focus on Tobie.team."
- **Quarantine**: If the user is unsure, files aren't deleted; they are moved to a _Quarantine_YYYY-MM-DD folder. This lowers the anxiety of "deleting something

important".[18]

- **Personal Integration**: "I added 'Review Tobie.team specs' to your calendar for 8 PM after the kids are in bed." This connects the organization task to the user's life context.

2.4 Deliverables: Mode A Specifications

2.4.1 File Classification Heuristics (Python Snippet)

Python

```
import os
import re

# Regex for common clutter patterns
PATTERNS = {
    "screenshot": r"Screenshot\s\d{4}-\d{2}-\d{2}",
    "installer": r".*\.(dmg|pkg|iso)$",
    "document": r".*\.(pdf|docx|txt|md)$",
    "image": r".*\.(jpg|png|heic)$",
    "code": r".*\.(py|js|html|css|json)$"
}

def classify_file(filename):
    """Returns a suggested category based on regex rules."""
    for category, pattern in PATTERNS.items():
        if re.search(pattern, filename, re.IGNORECASE):
            return category
    return "misc"
```

2.4.2 Organizing Checklist

- [ ] **Inventory**: Scan complete, file metadata (size, access time) indexed.
- [ ] **Plan**: Categorization map generated (Old Path -> New Path).
- [ ] **Review**: User approval received for micro-batch (max 10).
- [ ] **Snapshot**: State recorded in Undo DB (SQLite).
- [ ] **Execution**: Files moved securely.
- [ ] **Cleanup**: Empty directories removed (with explicit approval).

Section 3: Mode B (Trading Agent) Detailed Design

3.1 Execution Approach: API vs. Browser

Recommendation: API-First Approach.

For a system running 24/7 that manages financial risk, browser automation (Selenium/Playwright) is too brittle. UI changes by the broker can break the bot, and storing session cookies poses a security risk. We will strictly use authenticated **REST/WebSocket APIs**.

- **Crypto (Bitcoin)**: **Alpaca Markets**.
  - **Reasoning**: Best-in-class documentation, free paper trading environment, straightforward Python SDK (alpaca-trade-api), and real-time crypto data via WebSocket.[7]
- **Options (SPY)**: **Tradier**.
  - **Reasoning**: Tradier offers a free API for brokerage clients. It supports complex options chains and multi-leg orders (crucial for spreads). It also provides a Sandbox environment for paper trading options.[8]
- **Fallback**: If Tradier data coverage is insufficient (unlikely for SPY), integrate **ThetaData** for institutional-grade options feeds at low retail cost (~$40/mo).[19]

3.1.2 Operational Requirements

- **Uptime**: The agent runs as a daemon process (systemd or launchd on macOS).
- **Clock Sync**: Use NTP to ensure system time is synced to atomic clocks (crucial for candle closing times).
- **Heartbeat**: A "Dead Man's Switch" monitor that alerts the user (via Pushover or simple sound) if the data stream disconnects.

3.2 Data Sources & Market Connectivity

3.2.1 Data Provider Comparison

Provider	Asset Classes	Data Type	Latency	Cost	Terms/Notes
Tradier	Equities, Options	Real-time (Stream)	Low	Free (w/ brokerage)	Best for SPY Options. Free API access for account holders.[8]

Alpaca	Crypto, Equities	Real-time (Stream)	Low	Free (Basic) / $99 (Pro)	Best for Crypto/Bitcoin. Easy Python SDK.[7]
ThetaData	Options	Real-time (OPRA)	Ultra-Low	~$40/mo	Low-cost alternative for high-fidelity options data if Tradier lags.[19]
Polygon.io	All	Real-time	Low	~$30-200/mo	Excellent dev experience, but costlier than Tradier for retail.[20]

- **Bloomberg Terminal**: Not feasible or necessary. Costs ~$24k/year. We will rely on Tradier and Alpaca APIs which are fully compliant and sufficient for the defined scope.

3.2.2 News & Sentiment

- **Source**: **Financial Modeling Prep (FMP)** or **Alpha Vantage** (Free tiers available).
- **Integration**: The agent ingests headlines to check for high-impact keywords ("SEC," "Ban," "Rate Hike") to trigger a "No Trade" circuit breaker during volatility. This is not deep analysis but a safety filter.

3.3 Strategy Layer: "Trade Like Me"

3.3.1 Strategy Framework

The agent uses a **Decision Tree** approach rather than a "black box" neural network. This ensures explainability—you always know *why* it traded. The strategies are conservative and based on standard technicals.

- **Base Logic (Crypto - Trend Following)**:
  - **Trend Filter**: SMA (Simple Moving Average) 50 > SMA 200 (Golden Cross logic).[21]
  - **Volatility Filter**: Bollinger Band Width > Threshold (to avoid chop).[22] If the market is flat, the agent stays cash.

    - **Entry**: Price breaks above 20-period High.
    - **Exit**: Trailing Stop or Risk/Reward 1:2.
- **Base Logic (SPY Options - Defined Risk)**:
    - **Setup**: Vertical Credit Spreads (Bull Put Spread or Bear Call Spread).
    - **Delta Selection**: Sell 30 Delta, Buy 15 Delta (High probability of profit).
    - **Timing**: Enter only between 10:00 AM and 11:00 AM (avoid open volatility).
    - **T+1 Awareness**: Since May 2024, US equities and options settle in T+1.[23] This means capital recycles faster, but the agent must track "Unsettled Funds" accurately to avoid Good Faith Violations in a cash account.

3.3.2 The "My Rules" Overlay (Policy Engine)

To prevent overfitting and adhere to the user's specific constraints, the agent uses a strict Policy Engine that overrides any strategy signal.

- **Frequency Cap**: A counter tracks trades_last_5_days. If count >= 3, new entry signals are ignored.
- **Daily Loss Limit**: Hard-coded. If daily_pnl < -$50 (or user defined), the bot enters Sleep state until 00:00 UTC.
- **"Sleep Mode"**: No new trades after 3 PM (end of trading day) or during user-defined "Family Time" (retrieved from Mem0).

3.4 Phased Rollout Plan

1. **Phase 0: Simulation (Data Only)**
    - Agent connects to streams.
    - Logs "virtual trades" to a file. No orders sent.
    - **Gate**: 2 weeks of uptime with no crashes.
2. **Phase 1: Paper Trading (Crypto)**
    - Connects to Alpaca Paper API.
    - Executes Bitcoin trades with fake money.
    - **Gate**: 20 trades with positive expectancy and correct execution (no slippage errors).
3. **Phase 2: Paper Trading (Options)**
    - Connects to Tradier Sandbox.
    - Executes SPY spreads.
    - **Gate**: Validation of complex order types (multi-leg fills).
4. **Phase 3: Live "Tiny" Trading**
    - Real money, minimum position size (1 share or 1 cheap contract).
    - **HITL Required**: Agent proposes trade, User must press "Approve" key.
5. **Phase 4: Scaled Live (Restricted)**
    - Full sizing (within risk limits, max $3000 acct).
    - Autonomous execution allowed for *exits* (stops), but entries may still require approval.

Section 4: Permissions, Isolation, and Safety Controls

4.1 Threat Model & Mitigations

- **Threat**: **Hallucination**. The LLM decides to delete "System32" to clean up, or buy "All the Dogecoin" because of a misunderstood news headline.
  - **Mitigation**: The LLM *never* executes code directly. It outputs a structured intent (JSON). A deterministic Python "Executor" validates the JSON against a strict allowlist before running.
- **Threat**: **Fat Finger**. Loop error places 1000 orders in 1 second.
  - **Mitigation**: **Rate Limiter** middleware in the API wrapper. Max 1 order per minute.
- **Threat**: **Context Leak**. The Trader agent accesses personal files.
  - **Mitigation**: OS-level permissions (if feasible) or strict application-level sandboxing. The Trader module has no os.path access to ~/Desktop.

4.2 Tool Allowlists (Sandboxing)

Mode	Allowed Tools	Blocked Tools	Sandbox Path
A (Organizer)	os.rename, os.mkdir, shutil.move, read_metadata	os.remove (Hard Block), requests.post (No Web), subprocess	~/Desktop, ~/Downloads
B (Trader)	tradier_api.buy, alpaca_api.sell, get_quote	os.system, shutil.delete, browser.open	~/Trading_Logs

4.3 The "Panic" Kill Switch

A dedicated physical hotkey (e.g., Cmd + Esc) or distinct voice command ("CODE RED") immediately triggers:

1. **Process Termination**: Kills all agent Python processes.
2. **Order Cancellation**: Sends a CANCEL_ALL API request to the broker.
3. **Position Flattening**: (Optional, configurable) Closes all open positions at market.

Section 5: Voice Push-to-Talk & Hotkey Control

5.1 Offline-First Voice Design

To ensure privacy and speed, no voice data leaves the Mac.

- **Stack**: pynput (Hotkeys) + faster-whisper (Local STT).[24]
- **Workflow**:
  1. User holds Ctrl + Option + Space (mapped hotkey).
  2. sounddevice captures audio to a RAM buffer.[25]
  3. User releases keys.
  4. faster-whisper transcribes audio (on GPU/Metal if avail).
  5. Text is sent to the **Dispatcher**.

5.2 Reliability Checklist

- **VAD (Voice Activity Detection)**: Use silero-vad to trim silence and ensure only speech is processed. This prevents background noise from triggering the agent.[26]
- **Latency Goal**: < 2 seconds from key release to action initiation.
- **Feedback**: System plays a distinct "Listening" chirp (start) and "Processing" chime (stop). This auditory feedback is crucial for ADHD usability, confirming the system is attentive.

Section 6: Implementation Deliverables

6.1 Recommended Tech Stack

- **Language**: Python 3.10+ (Universal support for ML and FinTech libs).
- **Agent Framework**: **LangGraph**. It provides superior control over state loops and cycles compared to simple chains, crucial for the "Trade -> Monitor -> Update" loop.[1]
- **Local LLM**: **Ollama** running Llama 3.
- **Broker APIs**: **Tradier** (Options) + **Alpaca** (Crypto).
- **Vector Store**: **ChromaDB** (Local, file-based) managed via **Mem0**.

6.2 Skeleton Configuration (YAML)

YAML

```yaml
system:
  mode: "A" # Default start mode
  storage_path: "./data"
```

```yaml
dispatcher:
  hotkey: "<ctrl>+<alt>+<space>"
  model: "llama3"

mode_a_organizer:
  root_dirs:
  quarantine_dir: "~/Desktop/_Quarantine"
  safe_mode: true
  personality: "nosey_body_double"
  calendar_integration: true

mode_b_trader:
  broker: "paper_alpaca"
  risk_limits:
    max_daily_loss_usd: 50
    max_drawdown_pct: 2.0
    allowed_symbols:
  schedule:
    start: "09:30"
    end: "16:00"
```

6.3 Code Structure (Repo Layout)

```
/tobie-agent
├── /config              # YAML configs
├── /data                # SQLite DB, Vector Store, Logs
├── /core
│   ├── dispatcher.py    # Main router logic
│   ├── llm_engine.py    # Ollama wrapper
│   └── safety.py        # Kill switch & validation
├── /modes
│   ├── /organizer       # Mode A logic
│   │   ├── scanner.py
│   │   ├── actions.py   # Safe file ops
│   │   └── scheduler.py # Calendar integration
│   └── /trader          # Mode B logic
│       ├── broker.py    # API adapters
│       ├── strategy.py  # Signal logic
│       └── risk.py      # Circuit breakers
├── /voice
```

```
│   ├── listener.py        # Pynput + Sounddevice
│   └── transcriber.py     # Faster-Whisper
└── main.py                # Entry point
```

6.4 Test Plan

- **Unit Tests**: Mock file system operations (using pyfakefs) to ensure the Organizer never deletes root directories.
- **Integration Tests**: Run the Trader against historical data (Backtrader) to verify strategy logic.[27]
- **Dry Run**: Mode A runs in "Dry Run" mode by default, printing actions to a log without moving files until toggled off.

6.5 Day 1 Build Plan

1. **Environment Setup**: Install Python, Ollama, LangGraph.
2. **Dispatcher**: Build the simple text-based router.
3. **Mode A Skeleton**: Create the script that scans the desktop and lists files (read-only).
4. **Mode B Connection**: Connect to Alpaca Paper API and fetch the price of BTC.

Section 7: Future Outlook & Scalability

This architecture provides a scalable foundation. Future enhancements could include:

- **Multimodal Input**: Allowing the Organizer to "see" the screen (via screenshots) to better understand the context of the work being done.
- **Advanced RAG**: Ingesting all of the user's trading books into Mem0 to allow the Trading Agent to cite specific authors when proposing a trade ("This looks like a Wyckoff accumulation pattern...").
- **Mobile Companion**: A lightweight mobile app that feeds voice notes into the local system when the user is away from the Mac.

Clarifying Questions

To finalize the specific configurations for "Day 1", the following questions should be addressed:

1. **Broker Accounts**: Do you already have a funded Tradier or Alpaca account, or should the Phase 1 plan include "Account Setup" steps?
2. **Risk Parameters**: What is your specific dollar amount for "Daily Loss Limit" ($50? $100?) and "Max Drawdown" for the $3,000 account?
3. **File System**: Are there specific "No-Go" zones on your desktop (e.g., specific work

folders) that Mode A must explicitly ignore?
4. **Hardware**: Is the MacBook Air an M1, M2, or M3? (This affects the choice of the local LLM model size—7B vs 13B params).
5. **Voice Preference**: Do you prefer the voice agent to respond with **Text-to-Speech (TTS)** (talking back to you) or just **Action** (doing the task silently)?

End of Report

Works cited

1. Thinking in LangGraph - Docs by LangChain, accessed February 6, 2026, https://docs.langchain.com/oss/javascript/langgraph/thinking-in-langgraph
2. Thinking in LangGraph - Docs by LangChain, accessed February 6, 2026, https://docs.langchain.com/oss/python/langgraph/thinking-in-langgraph
3. Pattern I'm using to package local agent tools so they work across Node + Python - Reddit, accessed February 6, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1p1mx1a/pattern_im_using_to_package_local_agent_tools_so/
4. AI Memory for LLM Companions: Step-by-Step Guide - Mem0, accessed February 6, 2026, https://mem0.ai/blog/how-to-add-long-term-memory-to-ai-companions-a-step-by-step-guide
5. Mem0 - The Memory Layer for your AI Apps, accessed February 6, 2026, https://mem0.ai/
6. blakkd/faster-whisper-hotkey: Effortless Push-to-Talk Transcription, Anywhere. - GitHub, accessed February 6, 2026, https://github.com/blakkd/faster-whisper-hotkey
7. Unlimited Access, Real-time Market Data API - Alpaca, accessed February 6, 2026, https://alpaca.markets/data
8. Market Data - Tradier API, accessed February 6, 2026, https://documentation.tradier.com/brokerage-api/overview/market-data
9. Market Data - Tradier API, accessed February 6, 2026, https://docs.tradier.com/docs/market-data
10. A Beginner's Guide to Dynamic Routing in LangGraph with Command() | by Damilola Oyedunmade | AI Engineering BootCamp | Dec, 2025 | Medium, accessed February 6, 2026, https://medium.com/ai-engineering-bootcamp/a-beginners-guide-to-dynamic-routing-in-langgraph-with-command-2c8c0f3ef451
11. mem0ai/mem0: Universal memory layer for AI Agents - GitHub, accessed February 6, 2026, https://github.com/mem0ai/mem0
12. sqlite3 — DB-API 2.0 interface for SQLite databases — Python 3.14.3 documentation, accessed February 6, 2026, https://docs.python.org/3/library/sqlite3.html
13. ADHD-Friendly App Design: What to Look For (and What to Avoid) | Monster Math

Blog, accessed February 6, 2026, https://www.monstermath.app/blog/adhd-friendly-app-design-what-to-look-for-and-what-to-avoid
14. LLMs That Personalize Coaching for Agents With ADHD or Neurodivergence - Insight7 - Call Analytics & AI Coaching for Customer Teams, accessed February 6, 2026, https://insight7.io/llms-that-personalize-coaching-for-agents-with-adhd-or-neurodivergence/
15. shutil — High-level file operations — Python 3.14.3 documentation, accessed February 6, 2026, https://docs.python.org/3/library/shutil.html
16. Local History | PyCharm Documentation - JetBrains, accessed February 6, 2026, https://www.jetbrains.com/help/pycharm/local-history.html
17. Clean and efficient way to handle file system undo operations in a context manager, accessed February 6, 2026, https://stackoverflow.com/questions/54774667/clean-and-efficient-way-to-handle-file-system-undo-operations-in-a-context-manag
18. Automating Desktop Cleanup with Python — A Simple Script That Saves Me Every Day, accessed February 6, 2026, https://builder.aws.com/content/36NOLcWsfP40wGOlVjkrvFA1HYk/automating-desktop-cleanup-with-python-a-simple-script-that-saves-me-every-day
19. Pricing | Theta Data, accessed February 6, 2026, https://www.thetadata.net/subscribe
20. Cheapest real time / 15 Min delayed options data api (under $30/month) - Reddit, accessed February 6, 2026, https://www.reddit.com/r/algotrading/comments/1iw8bvx/cheapest_real_time_15_min_delayed_options_data/
21. Hello Algotrading! - Backtrader, accessed February 6, 2026, https://www.backtrader.com/home/helloalgotrading/
22. Indicators - Reference - Backtrader, accessed February 6, 2026, https://www.backtrader.com/docu/indautoref/
23. Understanding Settlement Cycles: What Does T+1 Mean for You? | FINRA.org, accessed February 6, 2026, https://www.finra.org/investors/insights/understanding-settlement-cycles
24. faster-whisper-hotkey - PyPI, accessed February 6, 2026, https://pypi.org/project/faster-whisper-hotkey/
25. Usage — python-sounddevice, version 0.4.1 - Read the Docs, accessed February 6, 2026, https://python-sounddevice.readthedocs.io/en/0.4.1/usage.html
26. Faster Whisper transcription with CTranslate2 - GitHub, accessed February 6, 2026, https://github.com/SYSTRAN/faster-whisper
27. Backtrader for Backtesting (Python) - A Complete Guide - AlgoTrading101 Blog, accessed February 6, 2026, https://algotrading101.com/learn/backtrader-for-backtesting/