Rostyslav Demyanchuk

Amber Agent

The Amber agent was something I worked on as a hobby. Most scripts are experimental. However 7 core scripts are functional. Please donate to my cash app: $Rosty7

The agent itself can respond like an llm but for those of you who want agentic functions there is a lot of them. Amber has a functions folder, where a function encyclopedia is stored. However, the contents if said encyclopedia are also here. Further down are more descriptions of other agents architecture and aother tools for the fterminal app. As well as datassette data app.

Description:

Agent F (Amber AI)

Agent F is a local-first macOS AI assistant powered by **Qwen 3 (8B)** running on Apple Silicon via **MLX**. It features a modular agentic architecture ("Amber") capable of routing natural language requests to specific Python scripts or JavaScript-based mini-apps.

The project is designed for rapid CLI iteration and deployment as a standalone macOS application via `py2app`.

🏗️ Architecture

The system is composed of four distinct layers:

1. **The Brain (Inference Layer)** - `ai.py`
* Loads model weights from `qwen.npz`.
* Handles the chat loop and manages the "Amber" persona.
* Strips internal `<think>` tags to ensure clean output.

2. **The Router (Agent Layer)** - `agent.py`
* Intercepts user inputs before they reach the LLM.

* Scans `agent-functions/` for available tools.
* Decides whether to reply conversationally or execute a tool (e.g., "Open calculator" `agentf-calc-launch.py`).

3. **The Tools (Dual Architecture)**
* **Python Scripts:** Located in `agent-functions/`. These handle OS-level tasks (File finding, App launching, Browser automation).
* **JS Mini-Apps:** Located in `apps/`. These are specialized tools (like the Calculator or Terminal) that launch in separate windows.

4. **The Body (GUI Layer)** - `gui.py`
* A PySide6 (Qt) wrapper that bootstraps the environment.
* Displays logs and manages the application lifecycle.

📂 Project Structure

```text
~/Developer/llm
├── agent.py                # Main agent routing logic & tool registry
├── ai.py                   # LLM inference loop (MLX)
├── gui.py                  # PySide6 macOS GUI entry point
├── build.sh                # Build script (cleans env, runs py2app)
├── setup.py                # py2app configuration & resource bundling
├── qwen.npz                # Model weights manifest
├── npz-weightmake-METAL.sh  # Script to generate qwen.npz from local model
├── agent-functions/        # 🛠️ Python Tool Scripts
│   ├── agent-browser-launch.py
│   ├── agentf-file-finder.py
│   ├── agentf-open-app.py
│   ├── use-browser.py
│   └── ...
└── apps/                   # 🖥️ JavaScript Mini-Apps
    ├── calculator/         # JS Calculator (currently under maintenance)
    └── terminal/           # JS Terminal
```

🚀 Capabilities

Amber is designed to be "helpful and precise." The agent currently supports the following capabilities via the `agent-functions` directory:

Command / Intent	Function	Description
`calclaunch`	**Calculator**	Launches the calculator mini-app (PySide6 + QtWebEngine).
`filefind`	**File Finder**	Fuzzy searches for files on the local system.
`browserlaunch`	**Browser**	Launches the custom AI Browser.
`openapp`	**App Launcher**	Opens native macOS applications (e.g., "Open Anki").
`maketxt`	**File I/O**	Creates UTF-8 text files from natural language.
`terminal`	**Terminal**	Executes system commands and custom tools.

🛠️ Setup & Installation

Prerequisites

* macOS (Apple Silicon recommended for MLX).
* Python 3.12 (managed via `.venv`).

1. Environment Setup

The project relies on a virtual environment. The build script checks for Python 3.12.

```bash
# Initialize venv (if not done automatically by build.sh)
./v-env.sh
source .venv/bin/activate

```

2. Model Weights

Amber uses Qwen 3 (MLX optimized). You must generate the `npz` manifest:

```bash
# Generates qwen.npz pointing to your local Qwen model
./npz-weightmake-METAL.sh
```

```

### 3. Running in CLI (Dev Mode)

For rapid iteration without rebuilding the `.app`:

```bash
# Assumes you have an alias or run python directly
python ai.py --weights qwen.npz --agent agent.py

```

4. Building the App

To package Agent F as a standalone macOS application (`dist/AgentF.app`):

```bash
./build.sh

```

Note: This script cleans the build artifacts, verifies the Python version, and executes `setup.py py2app`.

🐛 Troubleshooting & Known Issues

Calculator / JS Apps: Some JavaScript-based mini-apps (specifically in `apps/calculator`) are currently experiencing launch issues.
* **Permissions:** Ensure `agent.py` and helper scripts have execution permissions if running outside the python call.
* **Auto-Search:** If a terminal command fails, `agent.py` attempts an auto-search to find a fix.

📜 License
```

📘 The Amber Agent Grand Encyclopedia

Total Functions Listed: 129 **Version:** 1.0 (Master Compilation)

🚦 Status Legend

- 🟢 **WORKING / CORE:** Files currently uploaded and verified in your environment.
- 🔴 **KNOWN ISSUE:** Uploaded, but has a specific functional bug (detailed below).
- 🟠 **EXPERIMENTAL:** Not working. Requires `pip` dependencies or configuration to activate.

📚 Volume 1: Core System & Operating Environment

The foundational tools that allow the agent to exist, see, and interact with the host OS.

1. iMessage Sender (`agentf-imessage.py`)

- **Status:** 🟢 **CORE / WORKING**
- **Version:** 1.1 (AppleScript Bridge)

- **System Type:** Communication Bridge

📋 Executive Summary

Allows the agent to send blue-bubble iMessages or SMS text messages. Since Apple does not provide a public API for Messages on macOS, this tool uses the **AppleScript Bridge** (`osascript`). It instructs the Messages.app (in the background) to locate a "Buddy" matching the contact name or phone number and dispatch a text string.

⚙️ Technical Architecture

- **Language:** Python 3 + AppleScript.
- **Dependencies:** `osascript` (Native macOS tool).
- **Logic:**
    1. Constructs an AppleScript payload: `tell application "Messages" to send "{msg}" to buddy "{contact}"`.
    2. Executes via `subprocess`.
    3. Parses exit code to confirm delivery hand-off (note: cannot confirm *read* receipt, only *send* success).

🧩 Input/Output Schema

JSON

```
// Input (JSON)
{
  "contact": "Mom", // Name in Contacts.app OR Phone Number
  "message": "I'll be home in 10 mins."
}
```

🗣️ Prompt Examples

- "Text Mom that I'm on my way."
- "Send a message to 555-0199 saying hello."

1. Weather Reporter (`agentf-weather.py`)

- **Status:** 🟢 **CORE / WORKING**
- **Version:** 1.2 (Optimized)
- **System Type:** Data Retrieval / API Bridge

📋 Executive Summary

The primary meteorological interface for the agent. Unlike simple API wrappers, this tool employs a **Dual-Redundancy Architecture**. It attempts to fetch weather data from `wttr.in` (a console-optimized weather service) first. If that fails or provides insufficient data, it falls back to `Open-Meteo`, a free, high-precision open API. It features "Smart Location" logic: if the user does not specify a city, it automatically triangulates the host's IP address to determine the current location.

⚙️ Technical Architecture

- **Language:** Python 3 (Standard Library).
- **Dependencies:** `subprocess` (for cURL), `json`, `urllib`.
- **Data Flow:**
    1. **Sanitization:** Cleans user input (e.g., removes "current location" keywords).
    2. **Primary Request:** cURL request to `wttr.in/{location}?format=j1`.
    3. **Fallback Logic:** If Primary fails, queries `ip-api.com` for lat/lon, then queries `api.open-meteo.com`.
    4. **Parsing:** Extracts Condition, Temp (C/F), Humidity, Moon Phase, and 3-Day Forecast.
- **Error Handling:** Silent failover between providers; explicit error message if network is down.

🧩 Input/Output Schema

JSON

```
// Input (JSON)
{
  "location": "Paris, France", // Optional. If Omitted -> Auto-IP
  "full": true // Optional. Returns 3-day forecast if true.
}
```

🗣️ Prompt Examples

- "What is the weather right now?" (Triggers Auto-IP)
- "Check the forecast for Tokyo." (Specific Location)
- "Is it going to rain this week?" (Implies `full=true`)

2. Native Terminal (`agentf-terminal.py`)

- **Status:** 🟢 **CORE / WORKING**
- **Version:** 2.0 (GUI Persistent)
- **System Type:** Interactive Shell / GUI

📋 Executive Summary

A custom-built GUI terminal emulator designed specifically for Agent-User collaboration. Unlike a standard terminal (which is text-only and ephemeral), this tool launches a **Persistent PySide6 Window**. This allows the agent to execute shell commands in a visible environment where the user can watch the output in real-time. It supports session persistence, meaning variables set in one command remain available for the next (if implemented via the internal shell wrapper).

⚙️ Technical Architecture

- **Language:** Python 3 + PySide6 (Qt Framework).
- **Dependencies:** `PySide6`, `subprocess`, `os`.
- **Logic:**
    1. **Instance Check:** Checks if a Terminal Window is already open via a local socket/lockfile.
    2. **IPC Bridge:** If open, sends the command to the existing window via socket.
    3. **Launch:** If closed, spawns a new process rendering the UI.
    4. **Execution:** Runs commands via `subprocess.Popen` and streams `stdout` to the GUI text widget.

🧩 Input/Output Schema

JSON

```
// Input (JSON)
{
  "command": "ls -la" // The shell command to execute.
}
```

🗣️ Prompt Examples

- "Open the terminal."
- "List the files in the current directory."
- "Run a ping test to https://www.google.com/search?q=google.com."

3. FDatasette Viewer (`agentf-datasette.py`)

- **Status:** 🟢 **CORE / WORKING**
- **Version:** 1.0
- **System Type:** Database Visualization / Local Server

📋 Executive Summary

A specialized visualization tool for SQLite databases. It wraps the open-source `datasette` ecosystem into a desktop app. When triggered, it spins up a local web server (on localhost:8001) pointing to a specific database file (like your iMessage `chat.db` or browser history), then renders that server inside a native Qt Web Engine window. This gives the agent "Eyes" on raw data, allowing it to show you tables, filter rows, and run SQL queries visually.

⚙️ Technical Architecture

- **Language:** Python 3 + PySide6 + Datasette.
- **Dependencies:** `datasette` (must be installed via pip), `PySide6`.
- **Logic:**
    1. **Server Spin-up:** subprocess call to `datasette serve <db_file> -p 8001`.
    2. **GUI Launch:** Opens QWebEngineView pointing to `http://127.0.0.1:8001`.
    3. **Lifecycle:** Manages the background server process to ensure it dies when the window closes.

🧩 Input/Output Schema

JSON

```
// Input (JSON)
{
  "database": "/Users/name/Library/Messages/chat.db" //
Path to DB
}
```

🗣️ Prompt Examples

- "Open datasette” has to be told to the agent twice then when that is open you can launch data scripts such as the bluetooth scanner..
- "Show me my message history database."

4. Application Launcher (`agentf-open-app.py`)

- **Status:** 🟢 **CORE / WORKING**
- **Version:** 1.0
- **System Type:** OS Utility

📋 Executive Summary

The primary mechanism for the agent to affect the user's desktop environment. It utilizes the macOS Native `open`command to launch applications. It includes heuristic logic to handle fuzzy names (e.g., converting "chrome" to "Google Chrome.app") by checking the `/Applications` directory if a direct launch fails.

⚙️ Technical Architecture

- **Language:** Python 3.
- **Dependencies:** `subprocess` (calls `/usr/bin/open`).
- **Logic:**
    1. Try `open -a "AppName"`.
    2. If fail, append `.app` and retry.
    3. If fail, scan standard app directories for partial matches.

🧩 Input/Output Schema

JSON

```
// Input (JSON)
{
  "app_name": "Spotify"
}
```

🗣️ Prompt Examples

- "Open Spotify."
- "Launch Visual Studio Code."
- "Start Discord."

5. Weather Reporter (`agentf-weather.py`)

- **Status:** 🟢 **CORE / WORKING**
- **Version:** 1.2 (Optimized)
- **System Type:** Data Retrieval / API Bridge

📋 Executive Summary

The primary meteorological interface for the agent. Unlike simple API wrappers, this tool employs a **Dual-Redundancy Architecture**. It attempts to fetch weather data from `wttr.in` (a console-optimized weather service) first. If that fails or provides insufficient data, it falls back to `Open-Meteo`, a free, high-precision open API. It features "Smart Location" logic: if the user does not specify a city, it automatically triangulates the host's IP address to determine the current location.

⚙️ Technical Architecture

- **Language:** Python 3 (Standard Library).
- **Dependencies:** `subprocess` (for cURL), `json`, `urllib`.

- **Data Flow:**
    1. **Sanitization:** Cleans user input (e.g., removes "current location" keywords).
    2. **Primary Request:** cURL request to `wttr.in/{location}?format=j1`.
    3. **Fallback Logic:** If Primary fails, queries `ip-api.com` for lat/lon, then queries `api.open-meteo.com`.
    4. **Parsing:** Extracts Condition, Temp (C/F), Humidity, Moon Phase, and 3-Day Forecast.
- **Error Handling:** Silent failover between providers; explicit error message if network is down.

🧩 Input/Output Schema

JSON

```
// Input (JSON)
{
  "location": "Paris, France", // Optional. If Omitted -> Auto-IP
  "full": true // Optional. Returns 3-day forecast if true.
}
```

🗣️ Prompt Examples

- "What is the weather right now?" (Triggers Auto-IP)
- "Check the forecast for Tokyo." (Specific Location)
- "Is it going to rain this week?" (Implies `full=true`)

6. File Finder (`agentf-file-finder.py`)

- **Status:** 🟢 **CORE / WORKING**
- **Version:** 1.0
- **System Type:** File System Search

📋 Executive Summary

A high-speed search utility that bypasses slow Python recursive crawling. It leverages the macOS **Spotlight Index** via the `mdfind` command line tool. This allows the agent to find files across the entire hard drive in milliseconds, returning the absolute paths of the top matches.

⚙️ Technical Architecture

- **Language:** Python 3.
- **Dependencies:** `mdfind` (Native macOS tool).
- **Logic:**
    1. Executes `mdfind -name "{query}"`.
    2. Filters out system files or hidden directories (optional config).
    3. Returns the top 5-10 results to the Agent's context.

🧩 Input/Output Schema

JSON

```
// Input (JSON)
{
  "filename": "quarterly_report"
}
```

🗣️ Prompt Examples

- "Find that PDF about the merger."

- "Where did I save the budget spreadsheet?"

7. Amber Browser (`agent-browser-launch.py`)

- **Status:** 🟢 **CORE / WORKING**
- **Version:** 1.0
- **System Type:** Web Navigation

📋 Executive Summary

A dedicated web browser for the agent. While the agent can use `open` to launch Safari/Chrome, this tool launches a **Controlled QtWebEngine** instance. This is significant because it allows for future expansion where the agent could inject JavaScript, take screenshots, or scrape content directly from the window DOM, which is difficult in Chrome/Safari due to security sandbox restrictions.

⚙️ Technical Architecture

- **Language:** Python 3 + PySide6 (QtWebEngine).
- **Dependencies:** `PySide6`.
- **Logic:**
    1. Initializes a QApplication.
    2. Sets up a QWebEngineView.
    3. Loads the URL provided in arguments (or defaults to Google).
    4. Renders the window.

🧩 Input/Output Schema

JSON

```
// Input (JSON)
{
  "url": "https://news.ycombinator.com"
}
```

🗣️ Prompt Examples

- "Open the browser to Hacker News."
- "Surf the web."

8. Calculator (`agentf-calc-launch.py`)

- **Status:** 🟢 **CORE / WORKING**
- **Version:** 1.0
- **System Type:** Utility Widget

📋 Executive Summary

Launches a lightweight, aesthetic calculator. Instead of opening the heavy macOS Calculator app, this spins up a minimal web-based calculator (HTML/CSS/JS) inside a stripped-down Python window. This serves as a proof-of-concept for the agent's ability to launch "Micro-Apps" stored in its own library.

⚙️ Technical Architecture

- **Language:** Python 3 + PySide6.
- **Dependencies:** Requires a local `index.html` file in the `apps/calculator` directory.
- **Logic:** Loads a local file URI (`file://.../index.html`) into a frameless or minimal WebEngine window.

🧩 Input/Output Schema

JSON

```
// Input (JSON)
{
  "mode": "standard" // Reserved for future use
}
```

🗣️ Prompt Examples

- "Open the calculator."

9. Apple Notes (`agentf-notes.py`)

- **Status:** 🔴 **KNOWN ISSUE**
- **Issue:** The script successfully identifies the request and launches the Apple Notes application, but the AppleScript logic intended to *create a new note with specific text* fails to execute or is blocked by sandbox permissions, resulting in just the app opening blank.
- **System Type:** Productivity

📋 Executive Summary

Intended to allow the agent to dictate notes directly into the user's iCloud Notes. It uses AppleScript to target the "Notes" application, create a `new note`, and set its `body`.

⚙️ Technical Architecture

- **Language:** Python 3 + AppleScript.
- **Logic:**
  - `tell application "Notes" to make new note with properties {name: "Title", body: "Content"}`.
- **Fix Required:** Needs updated AppleScript syntax for macOS Sonoma/Sequoia or Accessibility permissions granting.

🧩 Input/Output Schema

JSON

```
// Input (JSON)
{
  "title": "Grocery List",
  "content": "Milk, Eggs, Bread"
}
```

🗣️ Prompt Examples

- "Create a note called Ideas."

📚 Volume 2: Productivity & Office (Experimental)

These tools exist in the codebase but require `pip` installation of dependencies to function.

10. Reminders (`agentf-reminders.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **System Type:** OS Integration
- **Executive Summary:** Bridges the gap between the Agent and the native macOS Reminders app. Unlike creating a text file, this pushes tasks into the Apple ecosystem, meaning they sync to your iPhone and Watch immediately.
- **Architecture:** Uses Python to execute an AppleScript payload: `tell application "Reminders" to make new reminder with properties {name:"...", body:"..."}`.
- **Schema:** `{"task": "Buy milk", "list": "Groceries"}`
- **Prompt:** "Add 'Buy Milk' to my grocery list."

11. Calendar (`agentf-calendar.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **System Type:** Data Retrieval
- **Executive Summary:** Gives the agent "temporal awareness." It queries the local calendar database to retrieve events for the current day or a specific date range.
- **Architecture:** Wraps the `icalbuddy` CLI tool (which must be installed via Homebrew) or uses raw AppleScript to parse the `Calendar.app` database.
- **Schema:** `{"date": "today"}`
- **Prompt:** "What is my first meeting today?"

12. Email Client (`agentf-email.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **System Type:** Communication Protocol
- **Executive Summary:** A headless email client. It decouples email from a GUI, allowing the agent to read unread inbox headers or dispatch emails programmatically via SMTP.
- **Architecture:** Leverages the `himalaya` CLI (a Rust-based CLI email client) for secure IMAP/SMTP handling, or standard Python `smtplib`.

- **Schema:** `{"to": "boss@corp.com", "subject": "Update", "body": "Done."}`
- **Prompt:** "Email the team that I am signing off."

13. Google Sheets (`agentf-sheets.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **System Type:** Cloud Data
- **Executive Summary:** Turns a Google Sheet into a database for the agent. It can append rows (logging expenses, tracking habits) or read specific cells.
- **Architecture:** Uses `google-api-python-client`. Requires a `credentials.json` Service Account key in the root directory.
- **Schema:** `{"action": "write", "spreadsheet_id": "...", "data": ["2024-01-01", "$50"]}`
- **Prompt:** "Log this expense to my budget sheet."

14. PDF Reader (`agentf-pdf.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **System Type:** Data Ingestion
- **Executive Summary:** Solves the "Black Box" problem of PDFs. Extracts raw text strings from binary PDF files so the LLM can analyze the content.
- **Architecture:** Uses `pypdf` or the native macOS `textutil` command line tool to convert PDF -> TXT in memory.
- **Schema:** `{"filepath": "/Users/me/docs/contract.pdf"}`
- **Prompt:** "Read and summarize this PDF."

15. OCR Scanner (`agentf-ocr.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **System Type:** Vision / Conversion

- **Executive Summary:** Optical Character Recognition. Allows the agent to "read" screenshots, memes, or scanned documents where text is embedded in pixels.
- **Architecture:** A wrapper for `pytesseract` (Google's Tesseract Engine). requires `brew install tesseract`.
- **Schema:** `{"image_path": "screenshot.png"}`
- **Prompt:** "Extract the error code from this screenshot."

16. Notion (`agentf-notion.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **System Type:** Knowledge Management
- **Executive Summary:** Integrates with the Notion API to append blocks to pages. Useful for building "Second Brain" workflows where the agent organizes your thoughts.
- **Architecture:** Uses the official `notion-client` Python library. Requires an Integration Token.
- **Schema:** `{"page_id": "...", "content": "TODO: Fix bug"}`
- **Prompt:** "Add this to my Notion dashboard."

17. Obsidian (`agentf-obsidian.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **System Type:** Local Knowledge
- **Executive Summary:** Manipulates a local Markdown vault. Specifically designed to append text to "Daily Notes" based on the current date (YYYY-MM-DD.md).
- **Architecture:** Standard Python File I/O. Safe and offline.
- **Schema:** `{"content": "- [ ] New Task"}`
- **Prompt:** "Log this meeting in my Obsidian daily note."

18. Bear Notes (`agentf-bear.py`)

- **Status:** 🟠 **EXPERIMENTAL**

- **System Type:** URI Scheme Automation
- **Executive Summary:** Creates notes in the Bear App. Unlike AppleScript tools, this uses the robust `x-callback-url` scheme, making it fast and reliable.
- **Architecture:** Python `webbrowser` module triggers `bear://x-callback-url/create?text=...`.
- **Schema:** `{"text": "Note content", "tags": "ideas"}`
- **Prompt:** "Save this idea to Bear."

19. 1Password (`agentf-1pass.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **System Type:** Security
- **Executive Summary:** Allows the agent to fetch secrets (API keys, passwords) dynamically, preventing hardcoded credentials in scripts.
- **Architecture:** Wraps the `op` CLI tool provided by 1Password. Requires user biometric auth (TouchID) to release keys.
- **Schema:** `{"item": "OpenAI API Key"}`
- **Prompt:** "Get my AWS key from 1Password."

📚 Volume 3: Communication & Web (Experimental)

20. BlueBubbles (`agentf-bluebubbles.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** API client for BlueBubbles, allowing Android users or headless servers to send iMessages via a relay.
- **Architecture:** JSON POST requests to a local or remote BlueBubbles server instance.

21. Telegram (`agentf-telegram.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Controls a Telegram Bot. Useful for sending notifications to the user when they are away from the computer.
- **Architecture:** Uses `python-telegram-bot` or direct HTTP requests to the Telegram Bot API.

22. Discord (`agentf-discord.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Posts messages to specific Discord channels via Webhooks. Great for logging system events or agent status updates.
- **Architecture:** `requests.post()` to a Discord Webhook URL.

23. Slack (`agentf-slack.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Enterprise communication. Posts messages to Slack channels.
- **Architecture:** Slack Webhook or Bot API.

24. Twilio (`agentf-twilio.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Programmable SMS. Allows the agent to send standard text messages to any phone number globally.
- **Architecture:** Uses the `twilio` Python library and SID/Auth Token.

25. CLI Browser (`agentf-browser.py`)

- **Status:** 🟠 **EXPERIMENTAL**

- **Executive Summary:** A lightweight, text-only browser. It fetches HTML, strips tags, and returns readable text.
- **Architecture:** Uses `requests` and `BeautifulSoup`. Can use DuckDuckGo HTML endpoint for searching.

26. Deep Research (`agentf-tavily.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** The "Pro" search. Uses the Tavily API, which is specifically built for LLM agents. It returns clean, parsed context rather than raw HTML, reducing hallucination.
- **Architecture:** Tavily Python SDK.

27. Wikipedia (`agentf-wiki.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Access to the world's knowledge. Fetches summaries or full pages.
- **Architecture:** `wikipedia` Python library.

28. WolframAlpha (`agentf-wolfram.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Computational Knowledge Engine. Solves math, physics, and logic queries that LLMs struggle with.
- **Architecture:** `wolframalpha` library. Requires AppID.

29. YouTube Intel (`agentf-youtube.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Extracts transcripts from videos. Allows the agent to "watch" a video by reading its dialogue.
- **Architecture:** `youtube-transcript-api`. No official API key required for public videos.

30. ArXiv (`agentf-arxiv.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Academic paper search. Accesses the ArXiv repository for STEM research.
- **Architecture:** `arxiv` Python library.

31. RSS Reader (`agentf-rss.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Monitors news feeds. Can return the latest headlines from specific tech or news sites.
- **Architecture:** `feedparser` library.

32. Finance (`agentf-finance.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Real-time market data. Fetches stock prices, P/E ratios, and company news.
- **Architecture:** `yfinance` (Yahoo Finance scraper).

33. Google Places (`agentf-google-places.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Geospatial search. Finds businesses, ratings, and addresses.
- **Architecture:** Google Places API (requires API Key).

34. Food Delivery (`agentf-delivery.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Generates deep-links to DoorDash, UberEats, or GrubHub search results for a specific food item.

- **Architecture:** URL construction (Launch logic).

35. Domino's Tracker (`agentf-dominos.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Interfaces with the unofficial Domino's Pizza API to track order status.
- **Architecture:** `pizzapy` library.

📚 Volume 5: Dev, Cloud & Hardware (Experimental)

36. GitHub (`agentf-github.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Interacts with repositories. Can read code, list issues, and check pull requests.
- **Architecture:** `PyGithub` library.

37. Jira (`agentf-jira.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Enterprise task management. Creates or queries tickets.
- **Architecture:** `jira` Python library.

38. Safe Coder (`agentf-coder.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** A utility for the agent to write code files to the disk safely. Includes a backup mechanism (renames existing files before overwriting) to prevent data loss.

- **Architecture:** Python File I/O with `shutil`.

39. Skill Maker (`agentf-skill-maker.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Meta-programming. Allows the agent to generate *new* `agentf-` tools based on user prompt, effectively expanding its own capabilities.
- **Architecture:** Text generation + File write.

40. Token Stats (`agentf-usage.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** An internal utility to calculate the cost savings of running local models vs GPT-4 based on token count.
- **Architecture:** Tokenizer logic.

41. Philips Hue (`agentf-hue.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Smart Home control. Toggles lights, changes colors/brightness.
- **Architecture:** `phue` library (Local Bridge API).

42. Eight Sleep (`agentf-sleep.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** IoT control for smart mattresses. Sets temperature.
- **Architecture:** Unofficial Eight Sleep API wrapper.

43. Cloud Audit (`agentf-scoutsuite.py`)

- **Status:** 🟠 **EXPERIMENTAL**

- **Executive Summary:** Security auditing. Wraps the ScoutSuite tool to scan AWS/Azure environments for misconfigurations.
- **Architecture:** `subprocess` call to `scout` CLI.

44. Game Launcher (`agentf-games.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Launches games via Steam or GOG Galaxy URL schemes (`steam://run/ID`).
- **Architecture:** `subprocess` call to `open`.

45. Battery Check (`agentf-battery.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Hardware monitoring. Reads power source, cycle count, and health.
- **Architecture:** Parses output of `pmset -g batt`.

46. System Stats (`agentf-system.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Performance monitoring. Real-time CPU, RAM, and Disk metrics.
- **Architecture:** `psutil` library.

📚 Volume 6: Bleeding Edge AI (Experimental)

47. AutoMode (`agentf-automode.py`)

- **Status:** 🟠 **EXPERIMENTAL**

- **Executive Summary:** The "Brain." A recursive ReAct (Reason + Act) loop. It takes a high-level goal, scans available tools, plans a sequence of actions, and executes them until the goal is met.
- **Architecture:** Loop using `litellm` for inference and dynamic tool loading.

48. Planner (`agentf-planner.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Generates a linear "flight plan" of commands to execute a task, then runs them sequentially. Good for deterministic tasks.
- **Architecture:** LLM Generation -> JSON List -> Sequential Execution.

49. Research Storm (`agentf-research-storm.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Inspired by Stanford STORM. A recursive research engine that searches, reads, generates new questions based on findings, searches again, and synthesizes a final long-form report.

50. Dev Loop (`agentf-dev-loop.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** A self-healing coding environment. It writes a script, attempts to run it, captures stderr (errors), feeds the error back to the LLM to generate a fix, and retries until success.

51. Vector Memory (`agentf-memory.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** RAG (Retrieval Augmented Generation). Stores text data as vector embeddings in a local database for long-term semantic recall.
- **Architecture:** `chromadb` + `sentence-transformers`.

52. GPT Vision (`agentf-vision.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Multi-modal analysis. Sends image data to OpenAI's Vision model to get textual descriptions.
- **Architecture:** Base64 encoding -> API Request.

53. Voice Clone (`agentf-eleven.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** High-fidelity Text-to-Speech. Can use cloned voices via ElevenLabs.
- **Architecture:** ElevenLabs API.

54. Gemini Fallback (`agentf-gemini.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Alternative Intelligence. Queries Google's Gemini Pro model if local models fail.
- **Architecture:** `google-generativeai` library.

55. Cloud STT (`agentf-stt-cloud.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Audio transcription using OpenAI Whisper (Cloud). High accuracy.
- **Architecture:** API Request.

56. Local Whisper (`agentf-whisper.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Audio transcription using Apple MLX (On-device). Privacy-focused.
- **Architecture:** `mlx_whisper` (optimized for Apple Silicon).

57. Image Gen (`agentf-image-pro.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Generates images from text prompts using DALL-E 3 or HuggingFace.
- **Architecture:** API Request.

📚 Volume 7: Automation & "Ghost" Tools (Experimental)

58. Stealth Browser (`agentf-ghost.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** The ultimate automation tool. Controls a Chrome browser instance that mimics human behavior to bypass bot detection (Cloudflare/recaptcha).
- **Architecture:** `DrissionPage` library.

59. Browser Auto (`agentf-browser-auto.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Standard automation for Safari or Edge using Selenium. Good for simple legacy tasks.
- **Architecture:** `selenium` webdriver.

60. Playwright (`agentf-playwright.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Modern, fast headless browsing. Excellent for rendering JS-heavy sites or capturing PDFs.
- **Architecture:** `playwright` library.

61. Browser Agent (`agentf-browser-use.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** An autonomous agent that navigates the web based on natural language instructions (e.g., "Buy me socks").
- **Architecture:** `browser-use` library + LangChain.

62. Autofill (`agentf-autofill.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Heuristic form filler. Navigates to a URL and attempts to identify signup fields (User/Pass/Email) to inject fake identity data.
- **Architecture:** `DrissionPage` + DOM Analysis.

63. Identity Faker (`agentf-identity.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Generates a consistent fake identity (Name, Address, Birthday, Username, Password) for use in automations.
- **Architecture:** Python `random` logic.

64. Temp Mail (`agentf-tempmail.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Interfaces with 1SecMail API to create disposable inboxes and read verification codes.
- **Architecture:** HTTP Requests.

65. Reddit Signup (`agentf-reddit-create.py`)

- **Status:** 🟠 **EXPERIMENTAL**

- **Executive Summary:** Targeted script for Reddit account creation. Handles form filling and waits for manual CAPTCHA solving.
- **Architecture:** `DrissionPage`.

66. X (Twitter) Free (`agentf-x-free.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Interacts with X.com without the paid API. Uses the web client's internal endpoints.
- **Architecture:** `twikit` library.

67. LinkedIn Connect (`agentf-linkedin.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Automates professional networking. Searches for keywords and sends connection requests.
- **Architecture:** `DrissionPage`.

68. Instagram Bot (`agentf-insta-bot.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Simulates a human user on Instagram to like posts on specific hashtags.
- **Architecture:** `DrissionPage`.

69. TikTok Upload (`agentf-tiktok.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **Executive Summary:** Automates video publishing to TikTok via the web interface.
- **Architecture:** `tiktok-uploader`.

📚 Volume 8: The Red Team Arsenal (Kali Linux Ports)

*This volume contains 48 specialized security tools. All are **EXPERIMENTAL** wrappers requiring the underlying binaries.*

70-82: Reconnaissance

- **70. Recon Basic (`agentf-recon.py`):** Whois/DNS lookup.
- **71. Nmap (`agentf-nmap.py`):** Port scanner/OS detection.
- **72. Shodan (`agentf-shodan.py`):** IoT/Device search engine.
- **73. Subdomains (`agentf-subdomains.py`):** Passive certificate log search.
- **74. Fierce (`agentf-fierce.py`):** Advanced DNS enumeration.
- **75. TheHarvester (`agentf-harvester.py`):** Email/Host OSINT gatherer.
- **76. Sherlock (`agentf-sherlock.py`):** Social media username hunter.
- **77. SpiderFoot (`agentf-spiderfoot.py`):** Automated OSINT orchestration.
- **78. Photon (`agentf-photon.py`):** High-speed crawler for data extraction.
- **79. Sublist3r (`agentf-sublist3r.py`):** Subdomain enumeration aggregator.
- **80. Dmitry (`agentf-dmitry.py`):** Deepmagic Info Gathering tool.
- **81. Amass (`agentf-amass.py`):** Attack surface mapping.
- **82. EyeWitness (`agentf-eyewitness.py`):** Visual web server reconnaissance.

83-90: Vulnerability & Exploitation

- **83. Nuclei (`agentf-nuclei.py`):** Template-based vulnerability scanner.
- **84. SQLMap (`agentf-sqlmap.py`):** Automatic SQL injection tool.
- **85. Commix (`agentf-commix.py`):** Command injection exploiter.

- **86. XSStrike (`agentf-xsstrike.py`):** Cross-Site Scripting scanner.
- **87. Wapiti (`agentf-wapiti.py`):** Web application vulnerability scanner.
- **88. Droopescan (`agentf-droopescan.py`):** CMS (Drupal/WP) scanner.
- **89. SearchSploit (`agentf-exploitdb.py`):** Offline exploit database search.
- **90. RouterSploit (`agentf-routersploit.py`):** Embedded device exploitation framework.

91-99: Network & Active Directory

- **91. Impacket (`agentf-impacket.py`):** Python network protocols (SecretsDump).
- **92. CrackMapExec (`agentf-cme.py`):** Post-exploitation tool for SMB/AD.
- **93. Responder (`agentf-responder.py`):** LLMNR/NBT-NS poisoner.
- **94. Mitm6 (`agentf-mitm6.py`):** IPv6 DNS takeover tool.
- **95. Enum4Linux (`agentf-enum4linux.py`):** SMB enumeration tool.
- **96. Scapy (`agentf-scapy.py`):** Interactive packet manipulation.
- **97. Wifite (`agentf-wifite.py`):** Automated wireless auditor.
- **98. MitmDump (`agentf-mitmdump.py`):** HTTP traffic interceptor/proxy.
- **99. Slowloris (`agentf-slowloris.py`):** Low-bandwidth DoS tool.

100-117: Forensics, Cracking & Misc

- **100. Decoder (`agentf-decoder.py`):** Multi-format string decoder.
- **101. HashID (`agentf-hashid.py`):** Hash type identifier.
- **102. Volatility (`agentf-volatility.py`):** RAM memory forensics.
- **103. Pypykatz (`agentf-pypykatz.py`):** Mimikatz implementation in Python.
- **104. OleVBA (`agentf-olevba.py`):** MS Office macro analyzer.
- **105. Binwalk (`agentf-binwalk.py`):** Firmware analysis tool.

- **106. IPA Analyze (`agentf-ipa-analyze.py`):** iOS app info extractor.
- **107. Androguard (`agentf-androguard.py`):** Android APK analyzer.
- **108. Mat2 (`agentf-mat2.py`):** Metadata removal tool.
- **109. StegCracker (`agentf-stegcracker.py`):** Steganography brute-forcer.
- **110. TruffleHog (`agentf-trufflehog.py`):** Git secret scanner.
- **111. H8Mail (`agentf-h8mail.py`):** Email breach checker.
- **112. CUPP (`agentf-cupp.py`):** Profiling password generator.
- **113. Patator (`agentf-patator.py`):** Multi-threaded brute-forcer.
- **114. SET (`agentf-set.py`):** Social-Engineer Toolkit.
- **115. Hydra (`agentf-hydra.py`):** Online login cracker.
- **116. Metasploit RPC (`agentf-msf-rpc.py`):** Automated MSF execution.
- **117. Gobuster (`agentf-gobuster.py`):** Directory/DNS brute-forcer.

📚 Volume 9: Advanced Cognitive Architecture (Experimental)

118. AutoGen Debate (`agentf-autogen-debate.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **System Type:** Multi-Agent Simulation
- **Executive Summary:** Simulates a Socratic debate between two AI personas to arrive at a better conclusion. The output of one agent is fed as input to the other.
- **Architecture:** `litellm` loop with distinct system prompts.
-

119. MemGPT Core (`agentf-memgpt.py`)

- **Status:** 🟠 **EXPERIMENTAL**
- **System Type:** Memory Management
- **Executive Summary:** Manages the agent's "Self." It reads/writes to a specific JSON file containing the user's bio and the agent's persona, distinct from the semantic vector memory. This ensures the agent never "forgets" who it is or who you are.
- **Architecture:** JSON File I/O.