"SarahMemory"

an AI-Bot Companion Platform

The Comprehensive Manual and Technical Guide Documentation

Version: 5.4.1/**Rev.Date**:03/22/2025 *© 2025 Brian Lee Baros. All Rights Reserved.*

Legal Notice

© 2025 Brian Lee Baros. All Rights Reserved.

WARNING: This Software is an experimental and evolving development of the SarahMemory AI-Bot Companion Platform. Although the Software is functional, unforeseen programming errors, unpredictable behavior, and potential system damage or instability may occur. The Software is provided "as is" without any warranties, express or implied. By using this Software, you assume all risks associated with its use, and the author assumes no responsibility for any adverse effects, malfunctions, or damage that might result from its use or misuse.

This Software and its associated documentation (collectively, the "Software") are the exclusive property of Brian Lee Baros. Unauthorized copying, distribution, modification, or any other use of the Software or any portion thereof, in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), without the prior express written permission of Brian Lee Baros is strictly prohibited.

The Software is protected by applicable copyright, trademark, and other intellectual property laws and international treaties. All rights not expressly granted herein are reserved by Brian Lee Baros. Any unauthorized use, reproduction, distribution, or modification of the Software may result in severe civil and/or criminal penalties and will be prosecuted to the fullest extent of the law.

By accessing and using the Software, you agree to be bound by these terms. If you do not agree to these terms, you are not authorized to use the Software.

All trademarks, service marks, logos, and trade names contained in the Software are the property of Brian Lee Baros or their respective owners, and no license or right to use any such trademark or trade name is granted or implied.

This notice constitutes the entire legal agreement concerning the Software and supersedes any prior or contemporaneous understandings, agreements, representations, or warranties, whether written or oral, regarding the Software.

Table of Contents

---

1. Introduction & Overview

1.1 Purpose of the Project

SarahMemory is conceived as a next-generation AI companion that revolutionizes human-computer interaction. Seamlessly integrating with Windows 10/11 (and later mobile platforms), it offers system optimization, creative assistance, automation, and emotional support. This manual is written in clear, accessible language so that even non-coders can appreciate the fusion of advanced AI techniques with robust engineering.

1.2 Vision and Importance

Inspired by films like *Tron* and *Her*, SarahMemory transforms science fiction into reality. It empowers users by managing routine tasks, optimizing system performance, and providing a personalized digital assistant that evolves over time. This project represents a pivotal step forward in productivity, automation, and creative expression, setting the stage for future autonomous AI solutions.

1.3 Personal Background and Motivation

I, Brian Lee Baros, having to overcome significant personal challenges, struggles and more as a single father, with a love for his children and enjoying imaginative thinking, from science fiction to documentaries, and biographies. These experiences fueled my determination to create an intelligent, compassionate tool that enhances efficiency and fosters meaningful connections. SarahMemory is my legacy—a flexible, modular platform built from scratch, capable of evolving autonomously even as a one-man band. It is a robust blueprint for future AI companions.

---

2. Project Concept and Vision

2.1 Core Objectives

- **Autonomous Learning and Adaptability:**
  SarahMemory continuously learns from user interactions via reinforcement learning and NLP, dynamically adjusting its behavior and responses.
- **Dynamic Resource Management:**
  The system monitors CPU, memory, GPU, and disk usage in real time, ensuring optimal performance even on aging hardware.
- **Self-Updating Mechanism:**
  With its innovative sandbox and self-updating "brain," SarahMemory tests new code in isolation and applies validated updates automatically.
- **Versatile Functionality:**
  It offers multi-modal interactions (voice, text, video, and visual avatar) and integrates with external systems, providing a strong base for future autonomous AI developments.

2.2 Overall Scope and Impact

SarahMemory is engineered to redefine personal and industrial productivity. Its intelligent, self-improving design is not only a personal assistant but also a dynamic platform capable of evolving with its users, attracting investment and inspiring further innovations in AI.

---

3. System Architecture and Resource Management

3.1 Hardware and Software Requirements

Minimum:

- **OS:** Windows 10/11
- **CPU:** Dual-core
- **RAM:** 8 GB (up to 4 GB allocated)
- **Storage:** 10 GB free space
- **GPU:** Integrated graphics
- **Peripherals:** Speakers, microphone, optional webcam
- **Internet:** For updates and online learning

Recommended:

- **CPU:** Quad-core or higher
- **RAM:** 16 GB+
- **Storage:** 20 GB+ on an NVMe SSD
- **GPU:** Dedicated graphics (e.g., NVIDIA GTX 1060)

3.2 Virtual Environment and Data Storage

SarahMemory runs in a sandboxed environment that optimizes resource allocation. Data is initially logged in text files and later stored in an incremental SQLite database with automated backups and a secure vault.

3.3 Detailed Scenario: Optimizing an Aging System

On older systems, SarahMemory dynamically minimizes resource usage while still providing full functionality. It prioritizes system responsiveness by throttling background learning during high-demand periods, using continuous performance monitoring with libraries like psutil.

---

4. Core Features and Functionalities

4.1 Autonomous Learning and Adaptive Behavior

- **Contextual Memory:**
  Every interaction is logged and analyzed using NLP techniques to build an intelligent, context-aware model.
- **User-Controlled Learning:**
  Users can manage learning settings via a robust GUI, ensuring complete control over data retention and privacy.

4.2 Dynamic Resource Management

- **Real-Time Monitoring:**
  The system continuously tracks CPU, memory, GPU, and disk usage and adapts resource allocation accordingly.
- **Virtual RAM Allocation:**
  When needed, additional virtual RAM is dynamically allocated to sustain performance under heavy load.

4.3 Automation, Macro Generation, and Self-Updating Capabilities

- **Task Automation:**
  Using automation libraries like PyAutoGUI, SarahMemory simulates user actions, reducing repetitive tasks.
- **Self-Updating Mechanism:**
  The system creates a virtual sandbox, gathers and tests new code, and, with user confirmation, updates its core modules—all while maintaining system integrity.

4.4 Security, Backup, and Recovery Mechanisms

- **Data Encryption:**
  Sensitive data is protected using robust encryption (Fernet).
- **Incremental Backups:**
  Automated backups safeguard the system, and a secure vault holds critical data.
- **Internal Security Scanning:**
  Integration with Windows Defender APIs ensures any new code is safe before deployment.

---

5. User Interface and Interaction

5.1 Graphical User Interface (GUI) Overview

The GUI provides an intuitive control panel using frameworks like Tkinter and PyQt. It allows users to manage chat, video, reminders, and system settings with ease, complete with status indicators and progress bars.

5.2 Voice, Text, and Visual Interaction

- **Speech-to-Text & Text-to-Speech:**
  Utilizing SpeechRecognition and pyttsx3, SarahMemory offers natural voice interaction.
- **Video Chat & File Sharing:**
  Enhanced video streaming, file transfer with acceptance prompts, and mute/video toggle options provide a rich communication experience.
- **Animated Avatar:**
  A transparent, interactive avatar mimics facial expressions and responds to voice commands, adding a personal touch to interactions.

---

6. Networking, Integration, and Remote Functionality

- **Peer-to-Peer Communication:**
  Socket programming enables real-time text and video chat with robust file sharing and mute/video toggling.
- **IoT & External Device Integration:**
  SarahMemory interfaces with devices like Arduino and Raspberry Pi to extend its automation capabilities.
- **Cloud Synchronization:**
  Dropbox integration ensures data is backed up and synchronized seamlessly.

---

7. Mobile Application Considerations

- **Optimized Design:**
  Future mobile versions will leverage Flutter or Electron for a touch-friendly, efficient interface.
- **Secure Data Sync:**
  Robust authentication and secure API connections will guarantee data privacy on mobile platforms.

---

8. Future Enhancements and Scalability

- **Cross-Platform Integration:**
  Designed to scale from Windows to mobile and beyond with seamless data portability.
- **Advanced Virtual Environments:**
  Future updates will include AI-generated 3D virtual worlds, enabling immersive digital command centers.
- **Customizable Personality & Emergency Features:**
  Users can fine-tune the AI's voice, appearance, and behavior, and activate emergency protocols if needed.
- **Plugin Architecture:**
  A modular plugin system will allow for easy addition and updating of features without altering the core codebase.

---

9. About the Author

Brian Lee Baros is a visionary software developer and a self-taught engineer with a unique background. As a former Navy serviceman and a single father, Brian has overcome significant challenges to pursue his passion for technology. His journey through hardship and perseverance led him to create SarahMemory—a groundbreaking AI companion designed to empower users, optimize performance, and bridge gaps in human interaction. Driven by personal experience and inspired by the potential of autonomous systems, Brian continues to innovate solo, proving that with determination, a one-man band can build an award-winning, transformative AI platform.

---

10. Project Files and Detailed Descriptions

Below is a list of key files Currently in the SarahMemory project at this version level, throughout the development of this project, file names, coding and purposes have been added, deleted and modified with each newer version , some files may not be compatible with older or newer versions of this software and along with their roles which may have changed:

- **SarahMemoryMain.py**
  Main entry point; integrates text chat, video chat, personality settings, cloud sync, research, and self-updating functions.
- **SarahMemoryAdaptive.py**
  Logs interactions and evolves the AI's personality based on user behavior.
- **SarahMemoryAdvCU.py**
  Provides context-aware responses to ensure intelligent, dynamic interactions.
- **SarahMemoryAiFunctions.py**
  Wraps API calls and integrates personality adjustments with contextual memory recall.
- **SarahMemoryAPI.py**
  Manages connections with external services such as Microsoft Graph and Azure OpenAI.
- **SarahMemoryAPIAccess.py**
  Facilitates interactions with external APIs (Google Calendar, Spotify, Weather, etc.).
- **SarahMemoryCognitiveServices.py**
  Integrates with Microsoft Cognitive Services for text and image analysis.
- **SarahMemoryDatabase.py**
  Manages the incremental SQLite database for logging interactions and backups.
- **SarahMemoryDL.py**
  Analyzes past conversations to detect patterns and suggest improvements.
- **SarahMemoryEncryption.py**
  Secures sensitive data using Fernet encryption.
- **SarahMemoryEsim.py**
  Simulates emotional responses to enrich interactions.
- **SarahMemoryExpressOut.py**
  Adds expressive phrases and emojis to messages.
- **SarahMemoryFacialRecognition.py**
  Uses OpenCV to capture and recognize faces for personalized greetings.
- **SarahMemoryFilesystem.py**
  Manages directories, file logging, deletion, and local backups.
- **SarahMemoryGUI.py**
  Provides a graphical user interface for various interactions.
- **SarahMemoryHi.py**
  Retrieves detailed system and hardware information.
- **SarahMemoryInitialization.py**
  Handles overall system initialization, including file system setup and encryption key generation.
- **SarahMemoryIntegration.py**
  Unifies all components into a single, autonomous system with a menu-driven interface.

- **SarahMemoryOAIS.py**
  Manages interactions with external AI systems.
- **SarahMemoryOptimization.py**
  Monitors system resources and triggers optimizations.
- **SarahMemoryParameters.py**
  Contains configuration settings, including API keys and system thresholds.
- **SarahMemoryPersonality.py**
  Manages the AI's personality settings and tailors responses accordingly.
- **SarahMemoryReminder.py**
  Consolidates reminder functionality using APScheduler for scheduling tasks.
- **SarahMemorySi.py**
  Lists installed software and provides controls for launching or closing applications.
- **SarahMemoryStartup.py**
  Manages Windows startup registration via the registry.
- **SarahMemorySync.py**
  Synchronizes local data with cloud storage using Dropbox.
- **SarahMemoryVault.py**
  Implements a secure, encrypted vault for sensitive data storage and backups.
- **SarahMemoryVoiceSynthesis.py**
  Converts text responses into speech using pyttsx3.
- **SarahMemoryAvatar.py**
  Manages avatar creation, facial expression mimicry, voice changes, and integration with external design software.
- **SarahMemoryVSM.py**
  Provides continuous voice input and system performance monitoring.
- **SarahMemoryGUIvideochat.py**
  Offers private chatrooms, video conferencing, file sharing, and enhanced interactive controls.
- **SarahMemorySynapes.py**
  The "brain" for self-updating and autonomous enhancements, managing sandbox testing and safe code modifications.
- **SarahMemoryResearch.py**
  Handles generic web research using asynchronous spider bots and search queries.
- **SarahMemorySoftwareResearch.py**
  Performs local research on installed software and operational guidelines.

11. Setup and Installation Instructions

11.1 Prerequisites

- **Operating System:** Windows 10 or Windows 11
- **Python Version:** 3.8 or higher
- **Required Libraries:**
  - pyttsx3, psutil, opencv-python, dropbox, APScheduler, SpeechRecognition, pyaudio, pyautogui, spacy, and others (as detailed in requirements.txt)

11.2 Download and Extraction

- Download the project source (ZIP file or Git repository).
- Extract the contents to a directory (e.g., `C:\SarahMemory\`).

11.3 Virtual Environment Setup (Recommended)

- Open Command Prompt and navigate to the project directory:

```
cd C:\SarahMemory\
```

- Create a virtual environment:

```
python -m venv venv
```

- Activate the virtual environment:

```
venv\Scripts\activate
```

11.4 Dependency Installation

- Install dependencies via the provided `requirements.txt`:

```
pip install -r requirements.txt
```

- If issues occur, try:

```
pip install --force-reinstall --no-cache-dir -r requirements.txt
```

- Download the spaCy English model:

```
python -m spacy download en_core_web_sm
```

11.5 Environment Variables and Cloud Sync Configuration

- Set environment variables for:
  - `GRAPH_API_TOKEN`
  - `OPENAI_ENDPOINT`
  - `OPENAI_API_KEY`
  - `CUSTOM_API_BASE_URL`
- For Dropbox integration, save your access token in: `C:\SarahMemory\data\cloud_token.txt`
- ***NEED HELP ON THIS? REFER TO THE TROUBLESHOOTING SECTION FOR A COMPLETE STEP-BY-STEP SETUP OF THE ENVIRONMENTAL VARIABLES***

11.6 Running the Project

- Open Command Prompt (with the virtual environment activated).
- Navigate to the project directory:

```
cd C:\SarahMemory\
```

- Launch the project:

```
python SarahMemoryMain.py
```

11.7 Using SarahMemory

- Follow the on-screen menu to access features:
  - Text chat, video chat, personality settings, system info, reminders, avatar management, research, self-update, etc.

11.8 Additional Configuration and Troubleshooting

- **Startup Registration:** Use `SarahMemoryStartup.py` to add SarahMemory to Windows startup.
- **Logging:** Check logs at `C:\SarahMemory\SarahMemory.log`.
- **Plugin Issues:**
  - If plugin installations fail:
    - Upgrade pip: `python -m pip install --upgrade pip`
    - Purge pip cache: `pip cache purge`
    - Force reinstall specific packages if necessary.

12. Troubleshooting Guidelines

12.1 Common Issues

- **Dependency Installation:**
  Verify that all libraries are installed.
  - **Step 1: Verify Your Python Installation**
  - **Check Python Version and Architecture:**
    1. Open a Command Prompt and type:
    2. python --version
       - Ensure it displays Python 3.13.x.
  - To verify architecture, run:
    1. python -c "import struct; print(struct.calcsize('P') * 8)"
       - This should output 64.
  - **Ensure Python and Scripts Directories Are in PATH:**
    1. Go to "Edit the system environment variables" in Windows.
       - Under Environment Variables, confirm that the paths for:
         - C:\Path\To\Python3.13\
         - C:\Path\To\Python3.13\Scripts\ are included in the PATH variable.

  - **Step 2: Set Up a Virtual Environment (Recommended)**
  - **Navigate to the Project Directory:**
    1. cd C:\SarahMemory\
  - **Create the Virtual Environment:**
    1. python -m venv venv
  - **Activate the Virtual Environment:**
    1. venv\Scripts\activate
       - Your command prompt should now show (venv) at the beginning of the line.

  - **Step 3: Install Dependencies**
    1. You can install all dependencies using the provided requirements.txt. If you face issues, follow the manual instructions below.
       - **A. Using requirements.txt:**
    2. **Standard Installation:**
       - pip install -r requirements.txt
         - **If Errors Occur (Force Reinstall):**
       - pip install --force-reinstall --no-cache-dir -r requirements.txt
  - **B. Manual Installation for Each Dependency**

1. Below are detailed instructions and common troubleshooting tips for each key dependency:
   - **pyttsx3 (Text-to-Speech)**
     - **Install:**
       - pip install pyttsx3==2.90
   - **Troubleshooting:**
     - If errors occur, try upgrading pip first:
       - python -m pip install --upgrade pip
     - Check if the package compiles correctly on Windows; precompiled wheels usually work well.
   - **psutil (System Monitoring)**
     - **Install:**
       - pip install psutil==5.9.5
     - **Troubleshooting:**
       - If you encounter build issues, ensure you have the proper Visual C++ Build Tools installed.
   - **opencv-python (Computer Vision)**
     - **Install:**
       - pip install opencv-python==4.8.0.76
     - **Troubleshooting:**
       - Sometimes, opencv-python may require the Visual C++ Redistributable. Download and install it from Microsoft's website.
   - **dropbox (Cloud Sync)**
     - **Install:**
       - pip install dropbox==11.34.0
     - **Troubleshooting:**
       - Verify your network connection and check if the package wheel is available for Python 3.13 on Windows.
   - **APScheduler (Task Scheduling)**
     - **Install:**
       - pip install APScheduler==3.9.1
   - **SpeechRecognition (Voice Input)**
     - **Install:**
       - pip install SpeechRecognition==3.8.1
   - **pyaudio (Audio Input)**
     - **Install:**
       - pip install pyaudio==0.2.11
     - **Troubleshooting:**
       - On Windows, if pip fails, download a precompiled wheel from Gohlke's unofficial binaries and install via:
     - pip install PyAudio-0.2.11-cp313-cp313-win_amd64.whl
   - **pyautogui (GUI Automation)**
     - **Install:**

    - pip install pyautogui==0.9.53
- **spacy (NLP Library)**
  - **Install:**
    - pip install spacy==3.5.1
  - **Download English Model:**
    - python -m spacy download en_core_web_sm
  - **Troubleshooting:**
  - If model download fails, check your network settings and try running the command as an administrator.
- **requests (HTTP Requests)**
  - **Install:**
    - pip install requests==2.28.1
- **Flask and python-dotenv (Optional, for web interfaces)**
  - **Install:**
    - pip install Flask==2.2.2 python-dotenv==0.21.0
- **openai (OpenAI API Integration)**
  - **Install:**
    - pip install openai==0.27.0
- **faiss-cpu (Vector Similarity Search, if used)**
  - **Install:**
    - pip install faiss-cpu==1.7.2
  - **Troubleshooting:**
    - This package can be tricky; if issues occur, check for precompiled wheels compatible with Python 3.13.
- **SQLAlchemy, loguru, click (For ORM, logging, CLI)**
  - **Install:**
  - pip install SQLAlchemy==1.4.46 loguru==0.6.0 click==8.1.3
- **aiohttp (Asynchronous HTTP Client)**
  - **Install:**
    - pip install aiohttp
- **beautifulsoup4 (HTML Parsing)**
  - **Install:**
    - pip install beautifulsoup4
- **googlesearch-python (Search Queries)**
  - **Install:**
    - pip install googlesearch-python
  - **Troubleshooting:**
    - If this package isn't found, search for "googlesearch" or use an alternative library.
- **PyQt5 (GUI for Avatar and Video Chat)**
  - **Install:**
  - pip install PyQt5

2. **Other Dependencies:**

- Ensure any additional modules referenced (e.g., Flask, openai, etc.) are installed with the version specified in your requirements.txt.

- **Step 4: Troubleshooting Techniques**
  1. **Upgrade Pip:**
     - python -m pip install --upgrade pip
  2. **Purge Pip Cache:**
     - pip cache purge
  3. **Force Reinstall:**
     - pip install --force-reinstall --no-cache-dir packagename
  4. **Manual Installation:**
     - If pip fails to install a dependency, visit Gohlke's website for precompiled Windows wheels.
- **Check PATH:**
  Ensure your Python installation and Scripts directory are in your system PATH.
- **Run as Administrator:**
  Some installations (especially for packages with native extensions) may require running Command Prompt as an administrator.

- **Step 5: Post-Installation Steps**
  1. **Verify Installations:**
     Run:
        - pip list
          - to check that all packages are installed.
  2. **Download Additional Models:**
     For spaCy, run:
        - python -m spacy download en_core_web_sm
- **Set Environment Variables:**
  Configure your environment variables (via Windows settings or a .env file) for:
  1. GRAPH_API_TOKEN
  2. OPENAI_ENDPOINT
  3. OPENAI_API_KEY
  4. CUSTOM_API_BASE_URL
- **Configure Cloud Sync:**
  Place your Dropbox access token in C:\SarahMemory\data\cloud_token.txt.

- **Step 6: Testing the Installation**

- **Run a Test Script:**
  Create a small Python file (e.g., test_install.py) with:

  - import spacy
  - import cv2
  - import pyttsx3
  - import requests
  - print("All dependencies imported successfully!")
  - Run it to ensure everything works:

    - python test_install.py
- **Check Logs:**
  Monitor the log file (if applicable) to ensure no errors occur during startup.

- **Step 7: Final Troubleshooting and Optimization**
- **If a Package Fails:**
  Try installing it without version numbers:
  1. pip install package_name
- **Manual Deletion:**
  If you encounter persistent errors, navigate to your Python site-packages directory (e.g., C:\Users\<USERNAME>\AppData\Local\Programs\Python\Python313\Lib\site-packages) and remove conflicting packages before reinstalling.
- **Consult Documentation:**
  For each dependency, refer to its official documentation for additional troubleshooting tips.
- **Community Forums:**
  Search Stack Overflow or the official GitHub issues page for common installation problems with Python 3.13 (64-bit).

- **Environment Variables: Troubleshooting:**

A Step-by-Step Guide to Configure Environment Variables

i. **Understand What You Need**

II. Your software requires several API keys and endpoints to work correctly:

III. **GRAPH_API_TOKEN:** Used for Microsoft Graph services.
   a. https://learn.microsoft.com/en-us/graph/auth-v2-service?tabs=http

IV. **OPENAI_ENDPOINT:** The URL endpoint for OpenAI services.
   a. https://chatgpt.com/g/WTF/oauth/callback think LINK JUST GIVES A 404 ERROR, but it can change or be whatever the End-User wants.

V. **OPENAI_API_KEY:** Your unique key to authenticate requests to OpenAI.

VI. **CUSTOM_API_BASE_URL:** A custom API URL for any additional service you might want to integrate. (If you don't use a custom API, you can leave this blank or use a default value provided by your organization.)

VII.

---

a. **2. How to Acquire Each API Key**

1. **GRAPH_API_TOKEN**

VIII. **Sign Up or Log In to Azure Portal:**

a. Visit [Microsoft Azure Portal](#) and sign in with your Microsoft account.

IX. **Register Your Application:**

a. In the Azure portal, navigate to **Azure Active Directory** > **App registrations**.

b. [Register an application - Microsoft Azure](#)

c. Click **New registration**, give your copy of this app a name the SarahMemory app is highly adjustable and fully customizable platform Ai-Bot (e.g., "Sarah, Skynet, Jarvis, It Doesn't Matter you can choose it "), and set the supported account types.

d. After registration, note down the **Application (client) ID** and **Directory (tenant) ID**.

X. **Configure API Permissions:**

a. In your app's registration, go to **API Permissions**.

b. Add the necessary permissions for Microsoft Graph (e.g., User.Read, Mail.Read, etc.), then click **Grant admin consent** if required.

XI. **Generate a Client Secret:**

a. Under **Certificates & secrets**, click **New client secret**.

b. Provide a description and set an expiration period, then click **Add**.

c. **Copy the client secret value** immediately (this is your token to be used as GRAPH_API_TOKEN).

XII. *For more details, see Microsoft's [Graph API documentation](#).*

XIII.

---

i. **B. OPENAI_ENDPOINT and OPENAI_API_KEY**

XIV. **Create an OpenAI Account:**

a. Go to [OpenAI's website](#) and sign up for an account if you haven't already.

XV. **Access the API Keys:**

a. Once logged in, navigate to the API section of your account dashboard.

b. Click on **View API keys** and then **Create new secret key** if you don't have one.

c. **Copy the API key** (this will be used as OPENAI_API_KEY).

XVI. **Determine the API Endpoint:**

a. The default endpoint is usually provided in OpenAI's API documentation (e.g., https://api.openai.com/v1/).

b. Use this URL as your OPENAI_ENDPOINT or modify it based on your project's needs.

XVII. *For further guidance, refer to the [OpenAI API documentation](#).*

XVIII.

---

i. **C. CUSTOM_API_BASE_URL**

XIX. **If You Have a Custom API:**

a. Contact your service provider or check your documentation for the base URL of your custom API.

b. Enter that URL in your configuration.

XX. **If You Don't Use a Custom API:**

a. You can either leave this variable blank or set it to a placeholder (e.g., https://yourcustomapi.example.com).

XXI.

---

a. **3. Configuring Environment Variables**

i. You can set these variables using one of two methods:

1. **Method 1: Using Windows System Environment Variables**

XXII. **Open Environment Variables Settings:**

a. Press **Win + S** and type “Environment Variables.”
b. Click **Edit the system environment variables.**
c. In the **System Properties** window, click the **Environment Variables…** button.

XXIII. **Add User Variables:**
a. Under **User variables** (for your account), click **New…**
b. Enter the variable name (e.g., GRAPH_API_TOKEN) and paste the corresponding value (the token you copied earlier).
c. Click **OK** and repeat this process for:
i. OPENAI_ENDPOINT
ii. OPENAI_API_KEY
iii. CUSTOM_API_BASE_URL (if applicable)

XXIV. **Apply Changes:**
a. Click **OK** to close all dialogs.
b. Restart any open command prompts or applications so they can read the updated variables.

1. Method 2: Creating a .env File

XXV. **Create the .env File:**
a. Open a text editor (e.g., Notepad).
b. Save a new file as a .env in the root directory of your SarahMemory project.

XXVI. **Add Your Variables:**
a. Enter your variables in the following format:

XXVII. GRAPH_API_TOKEN=your_graph_api_token_here
XXVIII. OPENAI_ENDPOINT=https://api.openai.com/v1/
XXIX. OPENAI_API_KEY=your_openai_api_key_here
XXX. CUSTOM_API_BASE_URL=https://yourcustomapi.example.com

a. Replace the placeholder text with your actual keys and endpoints.

XXXI. **Save the File:**
a. Save the file and ensure it is named .env (with no additional file extension).

XXXII. **Using the .env File in this Python Application:**
a. Use a library like python-dotenv to load these variables:

❖ from dotenv import load_dotenv
❖ import os
❖ load_dotenv() # This loads the variables from .env into your environment
❖ graph_api_token = os.getenv("GRAPH_API_TOKEN")
❖ openai_endpoint = os.getenv("OPENAI_ENDPOINT")
❖ openai_api_key = os.getenv("OPENAI_API_KEY")
❖ custom_api_base_url = os.getenv("CUSTOM_API_BASE_URL")

XXXIII. 6. Configure Cloud Synchronization (Optional)

XXXIV. **Dropbox Setup:**
Create a Dropbox app to obtain an access token.

XXXV. **Token Storage:**
Save your Dropbox access token in a file located at:
C:\SarahMemory\data\cloud_token.txt

i. Step-by-Step Instructions for Creating Your Dropbox Access Token

XXXVI. **Log Into Your Dropbox Account:**
a. Open your web browser and go to Dropbox.
b. Sign in with your existing Dropbox account.

XXXVII. **Navigate to the Dropbox Developers Section:**
a. Visit Dropbox Developers.

XXXVIII. **Create a New Dropbox App:**

a. Click on **"Create App"**.

b. Select the API type you need (typically **Scoped Access** is recommended).

c. Choose the type of access:

   i. **Full Dropbox:** Allows the app to access all files in your Dropbox.

   ii. **App Folder:** Restricts the app to a specific folder in your Dropbox.

d. Provide an appropriate name for your app (e.g., "SarahMemoryApp").

XXXIX. **Configure the App Settings:**

a. Once the app is created, you'll be taken to the app's settings page.

b. Under the **OAuth 2** section, you should see an option to generate an access token.

c. Click **"Generate access token"**.

d. Copy the generated access token.

XL. **Save the Access Token:**

a. Open a text editor (like Notepad).

b. Paste the access token into the file.

c. Save the file as cloud_token.txt in the directory C:\SarahMemory\data\.

   i. If the directory doesn't exist, create it manually.

      - Open Windows File Explorer
      - Go to the installed Directory Folder called SarahMemory and double click on it.
      - Right click in the area to create a NEW FOLDER
      - Rename the NEW FOLDER to "data" all lower case
      - Save the cloud.txt file created in Notepad to this folder

Trouble Shooting the File Paths:

Confirm paths in `SarahMemoryParameters.py` match your system configuration.

- **Go to the Run/Search Icon near the Start bar again and Type "Edit the System Environment Variable"**
- **Then Click on Environment Variable**
- **On the bottom Window where it says System Variable Scroll and find the word PATH then Click Edit**
- **On the Pop-up Window you should see any and all program Paths, be sure the <Drive>:\<wherever you installed python to directory>\PYTHON<version#>\ is in there and so is the same for the subdirectory <drive>:\PYTHON<version#>\Scripts are in there. If not Do that NOW or the install might not work.**
  Ensure all required variables are correctly set.

12.2 Any Additional Plugin Installation Troubleshooting or End Users Notes Can go here.

THIS PAGE WAS LEFT BANK ON PURPOSE

(Indiviual End-Users May Add Personal Experiences, Notes, Documentation and Pages in this Section)

12.3 Logging and Diagnostics

- Review `SarahMemory.log` for detailed error messages.
- Use the built-in system monitoring and diagnostic functions to identify performance issues.

---

13. Appendices and Additional Resources

13.1 Technical Diagrams and Flowcharts

- See Appendix A for a Mermaid diagram outlining SarahMemory's processing flow.

13.2 Change Logs and Troubleshooting Guides

- Detailed change logs are maintained with every version update.
- Troubleshooting guides include rollback procedures and incremental backup strategies.

13.3 References and Further Reading

- Microsoft Virtualization Documentation
- SQLite Documentation
- IEEE Xplore Digital Library
- PyAutoGUI Documentation
- SpeechRecognition Library on PyPI
- PyQt5 Documentation
- Arduino Official Site
- OWASP Mobile Security Project
- MIT Technology Review
- IEEE Spectrum

Overall Information Flow in SarahMemory

1. **User Input Reception:**
   When a user provides input (via text or voice), the system captures that input using its interaction modules (e.g., text chat, speech recognition).
   - **Example:** A user asks, "Tell me a joke."
2. **Contextual Processing & Memory Recall:**
   The input is forwarded to the contextual understanding module (e.g., **SarahMemoryAdvCU.py**) and adaptive learning modules (e.g., **SarahMemoryAdaptive.py**).
   - The system analyzes the input using natural language processing (NLP) techniques (via spaCy, for example) to determine context (is it a greeting, question, or command?) and logs the interaction.
   - Previous interactions stored in the incremental database are recalled to provide context.
   - **Example:** The system detects that the user enjoys humor from past interactions.
3. **Adaptive Behavior and Personality Evolution:**
   The adaptive module evaluates the current interaction patterns (logged in **SarahMemoryAdaptive.py**) and evolves its personality if needed.
   - This dynamic adjustment helps personalize the AI's responses over time.
   - **Example:** If humor is frequently requested, the personality may evolve to be more playful.
4. **Decision Making and Code Composition:**
   For tasks that require new functionality (or if a module is missing), the self-updating "brain" (in **SarahMemorySynapes.py**) along with research modules (**SarahMemoryResearch.py** and **SarahMemorySoftwareResearch.py**) are invoked:
   - The system creates a virtual sandbox, gathers data locally and online, and composes modular code to address the new request.
   - It then tests the new solution in isolation.
   - **Example:** The AI is asked to "make my avatar look like a mushroom." It will search locally for images or instructions, query the web if needed, and then compose a new module or update its avatar configuration accordingly.
5. **Response Generation and Output:**
   Once the context and any new code (if needed) have been processed, the AI synthesizes an appropriate response using its TTS module (**SarahMemoryVoiceSynthesis.py**) and visual output (for avatars, video chat, etc.).
   - **Example:** The system responds with a joke, delivered via both text and speech, and updates its adaptive memory for future interactions.
6. **Progressive Learning:**
   The system logs every interaction, which is later analyzed to continuously refine its models. Over time, as the database grows, the AI improves its contextual understanding and adaptability.
   - This creates a feedback loop that enhances the overall intelligence and responsiveness of the platform.

Flowchart Example (Mermaid Syntax)

You can use the following Mermaid diagram to visualize the logical flow:

```
flowchart TD
    A[User Input (Text/Voice)] --> B[Input Capture Module]
    B --> C[NLP Processing & Context Detection]
    C --> D[Memory Recall (Load Past Interactions)]
    D --> E[Adaptive Module: Log & Analyze Interaction]
    E --> F{New Task?}
    F -- Yes --> G[Invoke Self-Updating Brain]
    G --> H[Create Virtual Sandbox]
    H --> I[Gather Data Locally & from Web]
    I --> J[Compose Modular Code]
    J --> K[Sandbox Testing & Iterative Refinement]
    K --> L{Test Successful?}
    L -- Yes --> M[User Confirmation Prompt]
    M --> N{User Approves?}
    N -- Yes --> O[Apply Update & Restart System]
    N -- No --> P[Discard Sandbox Copy]
    L -- No --> Q[Retry (Up to 3 Times)]
    F -- No --> R[Generate Response Using Current Modules]
    R --> S[Text & TTS Response Output]
    S --> T[Update Interaction Logs & Learning Models]
    T --> A
```

❖ Detailed Explanation of the Flowchart

- ❖ **A → B:** The system receives input via text or voice.
- ❖ **B → C:** The input is processed using NLP to detect context and sentiment.
- ❖ **C → D:** The system recalls previous interactions from the database to add context.
- ❖ **D → E:** The adaptive learning module logs the current interaction and evaluates if adjustments are needed.
- ❖ **E → F:** The system checks whether the task is already handled or if new code is needed.
- ❖ **F (Yes) → G:** If a new task is detected, the self-updating “brain” (SarahMemorySynapes) is triggered.
- ❖ **G → H:** A virtual sandbox is created to safely test new code.
- ❖ **H → I:** The system gathers data using local resources and web research (via SarahMemoryResearch and SarahMemorySoftwareResearch).
- ❖ **I → J:** Modular code is composed based on gathered information.
- ❖ **J → K:** The new code is tested iteratively within the sandbox.
- ❖ **K → L:** The system checks whether the test is successful.
- ❖ **L (Yes) → M:** If successful, the system prompts the user for confirmation.
- ❖ **M → N:** The user confirms whether to apply the update.
- ❖ **N (Yes) → O:** If approved, the update is applied to the core system and a restart is initiated.
- ❖ **N (No) → P:** If rejected, the sandbox copy is discarded.
- ❖ **L (No) → Q:** If the test fails, the process is retried up to three times.
- ❖ **F (No) → R:** If the task is already handled, the AI generates a response using its existing modules.
- ❖ **R → S:** The response is output both as text and via TTS.
- ❖ **S → T:** The interaction is logged and used to update the learning models.
- ❖ **T → A:** The loop continues with new user input.

This concludes the current core foundation creation of the SarahMemory Ai-Bot Companion Platform in its current and functioning and operating state of progress that is created in this current build version 5.4.1

SarahMemory is more than an AI companion—it's a transformative, self-learning platform designed as a solid foundation for building autonomous AI-Bots. With its modular architecture, dynamic resource management, self-updating capabilities, and multi-modal interaction, SarahMemory represents the future of intelligent systems. This platform is designed to be both a functional assistant and a launchpad for further innovation, making it an exciting, award-worthy technology to own and to expand upon.