**AGI(Artificial General Intelligence)**

**Date**: October 26, 2023 (Update with current date)

**1. Introduction**

This document outlines the core functionalities of the AGI project. The project aims to create a conversational AI assistant accessible through a web interface. It embodies a Hybrid AI architecture, deliberately combining the strengths of:

* Subsymbolic AI: Represented primarily by the Large Language Model (LLM - Llama 3), which excels at natural language understanding, generation, pattern recognition, and learning from unstructured data.
* Symbolic AI: Represented by an Ontology-based Knowledge Base (Owlready2), which excels at structured knowledge representation, formal logic, consistency checking, and explicit reasoning.

This hybrid approach allows the AGI to leverage the LLM's fluency and broad understanding while using the symbolic component for reliable fact storage, structured reasoning, and potentially enforcing logical constraints.

**2. Hybrid Architecture Explained**

The AGI operates by integrating these two AI paradigms:

* Subsymbolic Core (LLM - Llama 3 via Groq):
  + Handles understanding user input in natural language.
  + Generates fluent, context-aware responses for general conversation.
  + Provides broad world knowledge and common-sense reasoning capabilities.
  + Can identify potential facts or intents within user input to be processed by the symbolic layer.
  + Uses recent conversational history (ConversationBufferWindowMemory) for dialogue flow.
* Symbolic Component (Knowledge Base - Owlready2):
  + Manages the Long-Term Memory (LTM) as a structured OWL ontology (knowledge\_base.owl).
  + Stores specific facts (individuals, properties) provided by the user via commands (e.g., "Remember my name is...") reliably and persistently.
  + Enables querying of these structured facts (e.g., "What is my name?").
  + Provides a foundation for future explicit logical reasoning (e.g., deduction based on class hierarchies, consistency checks) using ontology reasoners.
* Interaction:
  + LLM -> Symbolic: User input is processed. If a command to store/recall LTM facts is detected (by core\_logic.py), the relevant information is passed to the memory\_manager.py to interact with the Owlready2 ontology.
  + Symbolic -> LLM: A summary of key facts stored in the symbolic Knowledge Base (get\_ltm\_summary) is injected into the LLM's system prompt on each turn, providing it with crucial personalized context it wouldn't otherwise have reliably. Direct query results from the symbolic layer can also be used to generate specific factual responses.
* Other Components: Short-Term Memory (cache) and external services (weather, news, etc.) supplement this core hybrid interaction.

**3. Core User-Facing Functionalities**

(This section remains largely the same as before, but the underlying mechanisms are now better contextualized by Section 2)

* Conversational Interaction: Engage in natural language dialogue (primarily Subsymbolic/LLM).
* Information Retrieval: (Weather, News, Location - Uses external services, potentially informed by Symbolic LTM preferences).
* Task Execution: (Music Download, Email - Primarily procedural logic, potentially informed by LLM understanding).
* Memory Interaction:
  + Short-Term Recall/Storage/Forget (Procedural logic on volatile cache).
  + Long-Term Storage: Storing facts into the Symbolic Knowledge Base (Owlready2).
  + Long-Term Recall: Querying facts from the Symbolic Knowledge Base (Owlready2).
* Preference Management: (Stores preferences, potentially using Symbolic LTM for structure).
* Voice Input: (Frontend feature providing input to the Hybrid backend).
* Sentiment Awareness: (Utility function informing potential LLM response strategy).

**4. Memory System Detailed**

The AGI utilizes a multi-layered memory system supporting its Hybrid architecture:

* 4.1. Short-Term Memory (STM)
  + Technology: In-memory cachetools.TTLCache.
  + Role: Volatile storage for recent inputs, supporting specific recall commands. Independent of core Symbolic/Subsymbolic reasoning.
  + (Details remain the same: Duration, Capacity, Access, Indicator)
* 4.2. Long-Term Memory (LTM) / Knowledge Base (Symbolic AI)
  + Technology: Owlready2 Ontology stored persistently in knowledge\_base.owl.
  + Role: The primary Symbolic AI component for persistent, structured knowledge.
  + Purpose: Stores key-value facts formally, enabling reliable storage, retrieval, and potential logical inference.
  + (Details remain the same: Duration, Access, Integration via summary in prompt)
* 4.3. Conversational Memory (Part of Subsymbolic Interaction)
  + Technology: langchain.chains.conversation.memory.ConversationBufferWindowMemory.
  + Role: Provides immediate dialogue context to the Subsymbolic AI (LLM).
  + Purpose: Maintains conversational flow for the LLM.
  + (Details remain the same: Duration, Access)

**5. Backend Functionalities & Components (Python/Flask)**

The backend's modular structure reflects the Hybrid AI approach:

* app.py:
  + Main Flask application; handles web requests/responses.
  + Orchestrates initialization and shutdown.
  + Manages app\_state (holding both Symbolic and Subsymbolic components).
  + Initiates processing via core\_logic.py.
* core\_logic.py:
  + Central processing hub.
  + Performs intent recognition.
  + Routes requests: Directs memory commands to memory\_manager (Symbolic LTM / STM), service commands to services, and general conversation to the LLM chain (Subsymbolic).
* config.py: Central configuration (paths, keys, settings for both layers).
* storage.py:
  + Handles persistent file I/O.
  + Loads/Saves non-symbolic data (Prefs, History).
  + Loads/Saves the Symbolic Knowledge Base (knowledge\_base.owl).
* memory\_manager.py:
  + Interface for memory operations.
  + Includes functions for STM cache.
  + Includes functions (store\_ltm\_fact, recall\_ltm\_fact, get\_ltm\_summary) interacting with the Symbolic Ontology (Owlready2).
* services.py: External service interactions (Weather, News, TTS, etc.).
* utils.py: General utilities (logging, sentiment).
* LLM Integration (app.py, core\_logic.py): Manages interaction with the Subsymbolic LLM (Groq/Llama 3) via Langchain.
* Langchain Framework (app.py): Provides tools for building the Subsymbolic LLM interaction flow (memory, prompts, chains).
* Symbolic AI Component (storage.py, memory\_manager.py): Utilizes Owlready2 for managing the Symbolic Knowledge Base.

**6. Frontend Functionalities & Components (HTML/CSS/JavaScript)**

(This section remains the same as it describes the user interface, which interacts with the backend regardless of its internal hybrid structure)

* User Interface (Chat history, input area)
* Voice Input (Mic button, visualizer, SpeechRecognition API)
* Audio Output (TTS playback)
* Memory Indicators (Frontend display based on backend status flags)
* Asynchronous Communication (fetch API)
* Error Handling

**7. Technical Stack Summary**

* Backend: Python, Flask, Langchain, Langchain-Groq, Transformers (Hugging Face), PyTorch, VADER Sentiment, cachetools, Owlready2, yt-dlp, Requests, smtplib
* Frontend: HTML, CSS, JavaScript (Web Speech API, Web Audio API)
* Subsymbolic AI: LLM - Llama 3 (via Groq API)
* Symbolic AI: OWL Ontology Management via Owlready2
* TTS: VITS (via Hugging Face)
* Data Storage: JSON files (Prefs, History), OWL file (Symbolic LTM Knowledge Base)