**OpenAI Assistants AGI Architecture**

**Need to understand:**

* **What are the data types of everything being transferred and so how should it capture and transport the data?**
* **What are the BARE BONES OpenAI functions required**
* **What are the BARE BONES WORKFLOWS required for OpenAI data to move correctly**
* **—— 1) buttons for all the stuff**
* **—— 2) send message = send message to ThreadId and then createRun on thread for assistantId**
* **—— 3) call function from chat and add response to chat**
* **Bare bones for it to be able to update itself: 1. Ability to speak to assistant specialized as Project Operator(who speaks to Project Coordinator, who assembles chains and sends them to the Coding-specialist playwrite, who assembles the scripts for CodingChains and the pseudo-user sends back a green light to the Coordinator which indicates the script can be uploaded to the system and made available, who then uploads it and executes it, and sends a EXECUTING signal to the Project Operator) (NOT CLEAR IF PROJECT OPERATOR IS NEEDED. Probably only ever needs project coordinator)**
* **How do we then add Input=>STT=>TTS=>Output?**

**Main.py / index.js (create the application)**

**OpenAiClient.py/js (create the Client for the OpenAI endpoints)**

* **assistants**
* **1. Create assistant(with tool[code interpreter, retrieval, custom tools], json only mode options[yes{instructions}, no])**
* **2. List assistants**
* **3. Create thread**
* **4. CreateRunOnThread**
* **5. AddMessageToThread**
* **6. All data requirements**

**ChatClient.py/js**

* **1. /start endpoint for site to load chat front end connection and cause the chat to start with a default assistant**
* **2. /chat endpoint for site, when send message button is pushed, to cause data to be sent from front end and trigger: addMessageToThread(Data[Message], ThreadId) -> RunAssistant(AssistantId, ThreadId) -> AwaitResponse**

**Frontend.html**

* **chatWindow**
* **DropdownForListAssistants**
* **ButtonForCreateThread**
* **DropdownForThreadList**
* **DisplayThreadOnFrontend as chat history**
* **User chat messages blue**
* **Assistant run is black and white**
* **Ability to create new chatWindow**
* **Ability to drag and drop chatWindows on screen to position**
* **Ability to Input message on thread without run**
* **Ability to copy current messages and mirror output of one thread as messages input to another thread via chatWindow flow links from one to another one**
* **Ability to nest flows as tabs on windows**

**Coordinator**

* **instructions**
* **Tools: code interpreter, retrieval(workflows.csv), function call (workflowManager)**
* **Json mode: chainingChain**

**DivinePlaywrite**

* **instructions**
* **Tools: code interpreter, retrieval, function call:**
* **Json mode: assistantScriptCreationChain**
* **Creates Scripts -> sends to ProgenitorPantheonAutomobile**

**hardcodeAssistants.py**

* **create classes defined by the assistants and their scripts**
* **Basic: 0. Coordinator; 1. Progenitor(Profile Json Mode), Morph(Morph Json mode), Vajra(Vajra json mode); 2. Xpoll: XpQueen(TaskManagement from XpolEngine Json Mode), ExpertBee(Reasoning from Role-specific Chaining json mode[Morph, Vajra]); 3. DivinePlaywrite(AssistantScriptor);**
* **Advanced assistants: 4. CodingChain; 5. CrystalBall; 6. HoloInfoArchives; 7. GNO.SYS**

**assistantScripts.py**

**(Basic)**

* **Progenitor: function call to endpoint that runs assistant script to createRun for progenitor**
* **Pantheon Progenitor (Prog Automobile)**
* **Xpoll**
* **Xpoll Automobile**

**Advanced assistantScripts:**

* **CodingChain Automobile**
* **Create assistant from SDNA profile template (profile, chaining, database, everything integrated)**
* **CrystalBall Search**
* **HoloInfoArchivesSearch**
* **GNO.SYS: skillwave, journal entry(function call that updateFile)**

**How OPera GAN works:**

**OPera GAN refers to an agent topology in which:**

**There is an operator the user speaks to**

**The operator communicates with a coordinator using json mode: to infer a goal taxonomy from user to optimize coordinator’s chainingChain for zero shot**

**and then**

**passes that information to coordinator**

**The coordinator runs chainingChain on thread 2 and then returns a chain, which the pseudo-user triggers a function call for which sends that content back to thread 1 as a console message or throws error in same way after max rounds are reached in the DUO thread**

**The console message comes in as a MESSAGE NOT A RUN on thread1 and then the assistant (operator) only responds to it when the user sends another message**

**As for the work:**

**The coordinator makes chains which contain workflows of functions that cause:**

**Planning:**

**Planning the tasks for:{**

1. **Loading or Creating Execution assistants with custom instructions, tools for function call, json mode, and files for handling information regarding X as a synergy in a system**
2. **Flowing the assistants in an assistantsScript (planning Xpolls, progenitor automobiles, etc, workflows)**
3. **Constructing any special threads and logic for them**
4. **Running Xpolls (with optional DUOs for each expert, and the DUOs can be run in XpollDUO mode which makes the expertN into an internal team leader for themselves) and other workflows: pseudo-user runs scripts in the playwriting thread and reports the results back to playwrite until they complete the task**
5. **TaskExecutionChains: Crystal Ball Mapping -> IIC -> Expansion -> Loop for amplification going through chainTypes by creating one type of chain to then create another and combining them into a larger chain that involves their accomplishment (if chain is to get buyer avatar, fits in creating marketing campaign, fits in creating business, etc)**

**I want to build the version that excludes any functionalities termed “advanced” above.**