**User**

no.

it's this:

there is a flask app there is taskweaver there is openai assistants api

there is an openai assistant, every assistant has instructions. their instructions can include an SDNA PROFILE which is a highly descriptive ROLE with CHAINS for X. the complexprompt that has instructions can also have tools like code interpreter and retrieval or function calls. This openai assistant has a function call tool named SDNA Progenitor SDNA Progenitor is a promptchain that creates an SDNA PROFILE and a chain for the profile and then creates the assistant with the instructions including the SDNA PROFILE, chain, and an SDNA Progenitor function call tool so that it can make its own progeny (more specialized SDNA PROFILEs, chains).

there is a thread with message from a user this assistant can be called to the thread to do a run this message can trigger a function call on the run openai passes the function call to taskweaver which execs it on the flask app user interface has chat windows, user can add them user adds a new chat window user can connect flows between chat windows representing input and output this causes the messages from one thread to be sent to the next when the send message button is pressed and for a run to be executed when the message is received this run from the flow can include function calls that can include other promptchains

**Zip\_TaskWeaver**

Thank you for the clarification. Now that we have a more detailed framework, let's translate this into a clear structure:

### **System Components**

* **Flask App**: The core web application framework managing web server interactions and HTTP requests.
* **TaskWeaver**: A middleware platform operating within the Flask app responsible for complex task and workflow management.
* **OpenAI Assistants API**: The service integrated with the Flask app via TaskWeaver, providing AI-driven conversational capabilities, tools, and function execution.
* **SDNA PROFILE**: A domain-specific ROLE with an associated CHAIN, representing a set of instructions for tailored conversation handling.
* **Assistant Instance**: An entity created by OpenAI with specific instructions that include using an SDNA PROFILE for specialized domain handling and tools such as CodeInterpreter, Retrieval, and Function calls.
* **SDNA Progenitor**: A special type of function call enabled within an Assistant's instruction set to generate more specialized SDNA PROFILEs and CHAINs, effectively allowing Assistants to create sequences of tasks (progeny).

### **System Workflow**

* **Assistant Functionality**: Each Assistant, with its unique instructions and SDNA PROFILE, is invoked in response to user messages in threads.
* **Thread and Message Handling**: The user sends a message that initiates a thread, and the Flask app, utilizing TaskWeaver, calls upon the OpenAI Assistant to handle the run.
* **Function Calls**: Messages within the run can trigger function calls as defined by the Assistant's tools, causing OpenAI to pass these function call requests to TaskWeaver.
* **Task Execution in Flask App**: TaskWeaver processes the function call and executes the necessary actions within the Flask app.
* **User Interface Chat Windows**: The frontend allows users to add and manage chat windows corresponding to different threads or conversations.
* **Flow Customization**: Users can define flows between chat windows, setting up a series of input-output relationships between threads, effectively chaining conversations in a user-defined manner.
* **Message Passing and Run Execution**: As messages are passed through the connected flow, TaskWeaver ensures the runs are executed, incorporating messages and function calls in line with the created flow.
* **Automated Chain Reaction**: The defined flows between threads can trigger automated actions, with one node's output becoming another's input upon user interaction, creating a network of automated and interrelated conversations guided by SDNA PROFILEs and runs, facilitated by TaskWeaver and the OpenAI Assistants API.

In summary, this system envisions a highly modular, interconnected network of conversations and actions where the user has granular control over the dialogue flow, underpinned by a complex web of AI-driven logic and task execution capabilities leveraged from OpenAI and TaskWeaver, orchestrated within a secure Flask application environment.

**User**

so you see the issue i have is that we can create the progeny but we cant easily create the progeny and also assign them to work. we need a way to not only create the progeny in the promptchain but also to, upon creating them, create a thread with the context from the prior thread which they need, and to dictate how much of that context should be included in the new thread beforehand via a var in the function call. then we need the new assistant to use the same algorithm for doing work and creating progeny when it needs to do a ROLE drilldown.

what is a ROLE drilldown? in LLM based chat AIs, the context window can only handle so much complexity and it varies depending on how structured it is. so in order to properly manage this, we need to encapsulate ideas very strongly in the overall system, which is what ROLEs are for. in other words, the assistant can drilldown via its chat completions (this input2 can be deeper granularity than input1 was) but when it needs to accomplish work, it can only work on the same level it is at. that is, the ROLE can only include so many instructions, and the LLM can only follow so many instructions and can only deal with so much complexity, so it all has to be managed by a single algorithm. this algorithm is the Operatic Workflow. It is this: progeny doing their jobs until it's time for other progeny to do them. at that point Progeny become progenitors.

this makes esnse because no matter what, because of transformations, the object being transformed will change domains. when it changes domains, when it morphs, it has a morphism, and that morphism is a relationship in a process flow. so we can link tasks like this. so that is the theory of chaining, itself, in a way.

**Zip\_TaskWeaver**

The challenge you're facing revolves around creating a system that, upon encountering a task that requires more specific expertise or granularity than the current SDNA PROFILE can handle, will dynamically generate a more specialized profile, transfer the necessary context to it, and continue the work. This newly generated profile, or "progeny," becomes an "Assistant" that performs specialized tasks, or in your terms, a "ROLE drilldown."

This intricate process reflects the concept of "Operatic Workflow," where these specialized Assistants (progeny) can spawn further specialized entities (becoming progenitors next) to handle ever-increasing complexity and domain changes.

To implement such a system, we would need a robust workflow management and an effective strategy for context handling and transfer. Here's how the system flow would look like:

* **Context-Sensitive Progeny Creation and Assignment**: Develop functionality within TaskWeaver that analyzes the conversation context and determines when to generate a new SDNA PROFILE specialized for a new context or domain.
* **Thread Context Transfer**: Implement logic to carry over relevant context from a parent thread to a newly created thread when spawning a progeny SDNA PROFILE.
* **Context Limiting Variable**: Introduce a variable capturing context variables that dictates the extent of context transferred to the new progeny Assistant. This could control how much information is passed along to ensure the context window's constraints are respected, and complexity is managed.
* **Operatic Workflow Execution**: Build an algorithm into both TaskWeaver and the OpenAI Assistant that understands when a task has reached its complexity threshold and it's time to spawn progeny, effectively handling the "ROLE drilldown."
* **Domain Morphism Tracking**: Establish a system to track the morphisms, the changes in domains, and the processes that bring about those changes. This would allow the workflow to be aware of when and how to transfer tasks to the appropriate domain-specific PROFILE.
* **Automated Assistant Workflow**: Every Assistant, both the original and the progeny, will employ this Operatic Workflow algorithm to manage their tasks and to create further specialized progeny when necessary.

Building this system would likely require a centralized management service to keep track of all active Assistants, their relations, the tasks they are performing, and how those tasks map to different domains. This service must also manage the transfer of context and oversee the operations of the "progeny" in a secure and reliable manner.

Additionally, since the complexity and the instructions an LLM-based assistant can handle are limited by the context window, the system needs to intelligently manage the distribution of knowledge across various Assistants (SDNA PROFILEs). This could include chunking out tasks and maintaining a linked list or tree structure of Assistants and their progeny to ensure seamless operation and context handovers.

Implementing such an advanced and dynamic system introduces several challenges, including but not limited to:

* Managing state and context across hierarchical and potentially cyclic workflows.
* Ensuring data consistency and integrity throughout the flow.
* Safeguarding against infinite loops or unrestrained progeny generation.
* Securing all interactions and context transfers.
* Providing a user-friendly UI for monitoring and interacting with the live workflow.

The strategic deployment of this concept could yield a highly modular and scalable system capable of tackling tasks that require deep domain knowledge and specialization by leveraging the collaborative and hierarchical power of multiple AI Assistants, each proficient in its respective domain. This methodology could represent a paradigm shift in how conversational AI platforms are structured.

**User**

1 is taken care of by the user dynamically or by the chain in the profile statically. we dont need algorithms for it or 4 yet or 5.

**Zip\_TaskWeaver**

Understood. With the user dynamically creating SDNA PROFILES or the chain being predefined within a profile statically, you have a system in place that allows for the manual or predetermined generation of specialized profiles. Removing the necessity for the system to autonomously determine the requirements for new progeny significantly simplifies the process. Here's how the remaining elements could be structured:

### **Thread Context Transfer and Management**

Implement logic within TaskWeaver to initiate and manage the context transfer between threads when a new SDNA PROFILE is created. This includes:

* Collecting the necessary context data from the current Assistant thread.
* Instantiating a new Assistant thread with the context provided by the previous thread.
* Providing a mechanism to specify which parts of the context should be carried over to the new thread through a variable in the function call.

### **Function Calls Execution**

Incorporate the function calling capability of the OpenAI Assistant into TaskWeaver, allowing the backend to:

* Recognize function calls issued by the Assistant.
* Execute these function calls within the Flask app’s environment, which may involve triggering other services or APIs.
* Seamlessly transfer the results, along with the appropriate context, to the new or existing threads as defined by the workflow.

### **User Interface and Flow Customization**

Develop a frontend interface that allows users to:

* Add and manage multiple chat windows that represent different threads or conversations linked to specific SDNA PROFILEs.
* Define the logical flow and connections between chat windows, indicating data passage from one PROFILE to another.
* Visualize and control the execution pathways, potentially including thread spawning and function triggering.

### **Backend Integration and Workflow Execution**

Ensure that TaskWeaver coordinates with the Flask backend to:

* Handle the user-defined flows, executing the pathways as illustrated on the frontend.
* Manage real-time communication between threads, particularly when input from one thread’s message needs to be sent to another.

### **System Simplification and Focus**

By moving the responsibility of profile generation and chain definition to the user or predefining it within static chains, you can focus on the system's execution component:

* Efficient context handling, ensuring the right data is present in the right thread at the right time.
* Proper function execution, ensuring that the system can perform the tasks dictated by the SDNA PROFILEs and user-defined flows.
* User agency in workflow creation, enabling a flexible approach to defining and modifying how different parts of the system interact.

Focusing on these elements will streamline the implementation and provide users with powerful tools to leverage the TaskWeaver system in conjunction with OpenAI Assistant's capabilities.

**User**

so each window is a patch-in to an assistant on a thread, but if they connect input output flows, it means that the assistant run output from one thread will also be sent to another thread as an input from the user

“taskweaver:

thread\_bTPyD9z6SbhTakUAreFjxAI8

rivet:

thread\_JEjQYAA1SkY03wTJVfd2TTC1

“