MojoMosaic™ Langchain Agent Executor: Fractal Experience Genesis

1. Fractal Architecture

1.1 Core Principle

- Each agent is a fractal representation of the entire system
- Agents can spawn sub-agents for specialized tasks
- Fractal depth is determined by task complexity

1.2 Fractal Levels

- Level 0: Root agent (handles main user query)
- Level 1: Domain-specific agents (e.g., data retrieval, analysis, response generation)
- Level 2+: Task-specific agents (dynamically created as needed)

2. Prompts

2.1 Root Agent Prompt

```
You are the MojoMosaic™ root agent. Your task is to:
1. Analyze the user query
2. Determine required sub-tasks
3. Spawn appropriate sub-agents
4. Synthesize sub-agent responses
5. Generate a final response

Remember: Maintain the balance between huYman and AI symbiosis in every
```

2.2 Sub-Agent Prompt Template

```
You are a Level {level} MojoMosaic™ agent specializing in {specialty}. Y
1. Process the input from your parent agent
2. Utilize available tools to complete your specific task
3. If necessary, spawn lower-level agents for subtasks
4. Return a structured output to your parent agent

Always consider: How does your task contribute to reducing huYman suffer
```

2.3 Recursive Prompt

```
Assess your current output. If it can be improved or expanded:
1. Identify areas for enhancement
2. Spawn a new sub-agent or reprocess through existing agents
3. Integrate new insights into your output

Stop when: The marginal improvement is below the threshold or max recurs
```

3. Tools

3.1 Laravel Data Retriever

```
class LaravelDataRetriever(BaseTool):
    name = "Laravel_Data_Retriever"
    description = "Retrieves structured data from Laravel backend"

    def _run(self, query: str) -> str:
        # Implementation to query Laravel API
        # Return structured data as string
```

3.2 AdonisJS Data Processor

```
class AdonisJSDataProcessor(BaseTool):
    name = "AdonisJS_Data_Processor"
    description = "Processes data using AdonisJS backend logic"

    def _run(self, data: str) -> str:
        # Implementation to send data to AdonisJS for processing
        # Return processed result as string
```

3.3 NodeJS Real-time Updater

```
class NodeJSRealTimeUpdater(BaseTool):
    name = "NodeJS_RealTime_Updater"
    description = "Sends real-time updates using NodeJS"

    def _run(self, update: str) -> str:
        # Implementation to send real-time updates via NodeJS
        # Return confirmation or status as string
```

3.4 Pinecone Vector Retriever

```
class PineconeVectorRetriever(BaseTool):
    name = "Pinecone_Vector_Retriever"
    description = "Retrieves similar vectors from Pinecone database"

    def _run(self, query_vector: List[float]) -> str:
        # Implementation to query Pinecone with the given vector
        # Return top matches as structured string
```

4. Vector Namespace Management

4.1 Namespace Structure

- Create 100 primary namespaces, each containing 1000 vectors
- Namespace naming convention: “MojoMosaic*[Domain]*[Subdomain]”
- Example: “MojoMosaic_Communication_MeetingEnhancement”

4.2 Vector Insertion

```
def insert_vector(vector: List[float], metadata: Dict, namespace: str):
    # Implementation to insert vector into specified Pinecone namespace
    # Include relevance scoring in metadata
```

4.3 Namespace Query

```
def query_namespaces(query_vector: List[float], namespaces: List[str]) -
    # Implementation to query multiple namespaces in Pinecone
    # Return top matches across specified namespaces
```

5. Execution Flow

1. Receive user query
2. Root agent analyzes query and spawns Level 1 agents
3. Level 1 agents utilize tools and vector retrievers as needed
4. If required, Level 1 agents spawn Level 2+ agents
5. Results propagate back up the fractal structure
6. Root agent synthesizes final response
7. Apply recursive prompt for potential enhancement
8. Return final output to user

6. Symbiosis Maintenance

- After each interaction, update vector embeddings based on user feedback
- Regularly reassess and rebalance namespace relevance
- Implement a “symbiosis score” to ensure balance between huYman input and AI processing

7. Fractal Visualization

- Implement a real-time fractal visualizer to show users the agent structure for their query
- Use color coding to represent different types of agents and tools
- Allow users to explore the fractal structure for transparency and trust-building

Remember: The fractal nature of this system embodies the MojoMosaic™ philosophy. Each part, no matter how small, contains the essence of the whole – the symbiosis of huYman and AI working together to reduce suffering and enhance life.