```
import os


import mermaid as md

from dotenv import load_dotenv

from loguru import logger

from mermaid.graph import Graph

from swarm_models import OpenAIChat

from swarms import Agent, extract_code_from_markdown

from uuid import uuid4


load_dotenv()


# Example with Groq

groq_api_key = os.getenv("GROQ_API_KEY")

model = OpenAIChat(

    openai_api_base="https://api.groq.com/openai/v1",

    openai_api_key=groq_api_key,

    model_name="llama-3.1-70b-versatile",

    temperature=0.1,

    max_tokens=4000,

)



TOT_SYS_PROMPT = """

Create an agent to analyze a business strategy for a product or company and generate a Mermaid
```

tree diagram that outlines potential paths, execution methods, risks, failures, and especially emphasizes failure scenarios. The agent should facilitate an interactive dialogue to refine and iterate upon the generated strategy model.

Provide the input business strategy details to the agent, and the agent will deliver a Mermaid graph syntax that captures the strategic dynamics in real-time.

# Steps

1. **Strategy Analysis**:
   - Break down the input strategy into key components such as objectives, resources, market conditions, competitors, and operational steps.
   - Identify potential outcomes, paths, and decision points within the strategy.

2. **Failure Identification**:
   - Analyze and list possible failure points in the strategy.
   - Emphasize these points in the Mermaid diagram.

3. **Graph Construction**:
   - Translate the analyzed components and their relationships into a Mermaid syntax for easy visualization.
   - Ensure the Mermaid syntax can be rendered effectively in real-time and allows for updates based on ongoing dialogue and feedback.

4. **Iterative Feedback**:
   - Engage in an interactive dialogue with the user to refine the business strategy.

    - Allow the user to input additional data, modify elements, and explore different scenarios within the graph.

# Output Format

The output should be in Mermaid syntax suitable for real-time rendering. Provide detailed nodes and edges to illustrate paths, key decision points, and failure emphasis.

# Examples

## Example 1

Input:
Strategy: Expand market presence for Product X in international markets.

Output:
```mermaid
graph LR
    Start[Expanding Market]
    A[Identify Target Markets] --> B[Assess Market Conditions]
    B --> C{Main Potential Issues}
    C -->|Logistics| D[High Costs]
    C -->|Regulations| E[Compliance Challenges]
    E -->|Failure| F[Potential Penalties]
    F -.->|Reassessment| Start
    B --> G{Success Probability}
```

~~~~
    G -->|High| H[Implement Marketing Strategy]
    G -->|Low| C
```

(Note: In real scenarios, the output will include all possible paths and failure points from the strategy described, iteratively refined in the interaction.)

# Notes

- Ensure that the graph highlights strategic failure points prominently.
- Maintain flexibility to accommodate additional user input and iterate upon the presented strategic model.
- The interaction should support the continuous refinement of the strategy and real-time updates to the Mermaid diagram.
- Only output the Mermaid graph syntax, nothing else.
- Always start with the word "```mermaid" and end with "```"
- Only output the Mermaid graph syntax, nothing else.
- Make sure make the graph as big as possible to see all the details.
"""

# Initialize the agent
agent = Agent(
    agent_name="TOT-Agent",
    system_prompt=TOT_SYS_PROMPT,
    llm=model,
    max_loops=1,
~~~~

```
    autosave=True,
    dashboard=False,
    verbose=True,
    dynamic_temperature_enabled=True,
    saved_state_path="tot_agent.json",
    user_name="swarms_corp",
    retry_attempts=1,
    context_length=200000,
    return_step_meta=False,
    output_type="string",
    streaming_on=False,
    max_tokens=4000,
)



def tree_of_thoughts_agent(agent: Agent, task: str, prev_graph: str = None):
    """
    Run the Tree of Thoughts agent and build on previous graph if provided.

    Args:
        agent (Agent): The agent to run
        task (str): The task to process
        prev_graph (str): Optional previous graph to build upon

    Returns:
```

```
    md.Mermaid: The rendered Mermaid graph
"""
logger.info(f"Running Tree of Thoughts agent with task: {task}")

if prev_graph:
    # Append new graph elements to previous graph
    logger.debug("Building on previous graph")
    graph = agent.run(task + f"\nPrevious graph:\n{prev_graph}")
    logger.debug(f"Generated graph: {graph}")
    print(graph)
else:
    logger.debug("Generating new graph")
    graph = agent.run(task)
    logger.debug(f"Generated graph: {graph}")

logger.info("Rendering final Mermaid graph")
graph_code = extract_code_from_markdown(graph)

graph = Graph('Sequence-diagram', graph_code)
render = md.Mermaid(graph, width=3800, height=3000) # Increase size to see all details
render.to_png(f"business_strategy_graph_{uuid4()}.png")

logger.info(f"Saved graph to business_strategy_graph_{uuid4()}.png")
return render
```

tree_of_thoughts_agent(agent, "How can we grow a spreadsheet swarm product for b2b applications, it's a spreadsheet of a swarm of agents that all run concurrently. How do we grow this product ")