BrandMate: A Multi-Agent AI Solution for LinkedIn Influencers

Aruni Saxena, Paarth Patel, Rishi Barapatre, Anandita Saolapurkar

May 26, 2025

Abstract

LinkedIn influencers struggle with inconsistent brand messaging, keeping up with trends, gaining performance insights, and burnout from manual content creation. `BrandMate`, developed by Aruni Saxena, Paarth Patel, Rishi Barapatre, and Anandita Saolapurkar, is a multi-agent AI system built with LangGraph to address these challenges. It delivers SEO-optimized posts, trend-aware hashtags, automated scheduling, performance tracking, and a feedback dashboard. This document details the problem, solution, deliverables, team contributions, development reflections, project resources, and a deep dive into the Reinforcement Learning (RL) agent's Q-learning mechanism, noting the challenge of RL deployment.

1 Problem Statement

LinkedIn influencers face:

- **Inconsistent Brand Messaging**: Varying tone and style dilute brand identity.
- **Difficulty Keeping Up with Trends**: Rapidly changing trends require constant research.
- **Poor Post Performance Insight**: Limited data hinders strategic improvement.
- **Burnout from Manual Content Creation**: Manual post creation causes fatigue.

2 Multi-Agent AI Solution

`BrandMate` addresses these challenges:

- **Personalized Tone & Trend-Aware Writing**: Extracts tones and integrates trending keywords/hashtags.
- **Specialized Agents Handle Distinct Tasks**: Modular agents streamline content creation.
- **Optimized Content Through Feedback Loops**: RL agent refines strategies.
- **Content Evolves with Performance Data**: Adapts based on engagement data.

Table 1: BrandMate Agent Roles

Agent	Role
`user_interaction_agent`	Extracts tone, target audience, and goal
`brand_identity_agent`	Defines style guide (tone, sentence length, vocabulary)
`content_strategist_agent`	Selects a relevant topic
`seo_agent`	Fetches keywords and hashtags via SerpAPI
`content_generator_agent`	Generates a raw LinkedIn post
`post_editor_agent`	Refines the post and adds branding hashtags
`publishing_agent`	Simulates posting and generates metrics
`rl_feedback_agent`	Updates Q-table and suggests strategies

3 Proposed Solution: BrandMate Architecture

`BrandMate` uses LangGraph to orchestrate eight agents, updating a `BrandMateState` dictionary in a directed acyclic graph (DAG) executed via `app.invoke`. Agent roles are:

The workflow uses:

- ▷ `workflow.add_node(node_id, function)`: Registers agents.
- ▷ `workflow.add_edge(from_node, to_node)`: Defines sequence.
- ▷ `workflow.set_entry_point("user_interaction")`: Sets starting node.
- ▷ `workflow.compile()`: Creates executable `app`.
- ▷ `app.invoke(initial_state)`: Runs the graph.

3.1 Streamlit Deployment Code

The following LangGraph code, integrated with Streamlit, demonstrates the `BrandMate` workflow. This code powers the deployment interface, with screenshots available in the project repository:

```python
import streamlit as st
from langgraph.graph import StateGraph, END
from typing import TypedDict, Dict, Any

class BrandMateState(TypedDict):
    user_input: str
    tone: str
    target: str
    goal: str
    style_guide: Dict[str, str]
    topic: str
    seo_package: Dict[str, Any]
    post: str
    metrics: Dict[str, Any]
    feedback: str
    q_table: Dict[str, Any]

# Placeholder agent functions
def user_interaction_agent(state: BrandMateState) -> BrandMateState:
    state["tone"] = "emotional"
    state["target"] = "women startup founders"
    state["goal"] = "grow followers"
    return state
# ... (other agent functions defined similarly)

# Initialize workflow
workflow = StateGraph(BrandMateState)
workflow.add_node("user_interaction", user_interaction_agent)
workflow.add_node("brand_identity", brand_identity_agent)
workflow.add_node("content_strategist", content_strategist_agent)
workflow.add_node("seo", seo_agent)
workflow.add_node("content_generator", content_generator_agent)
workflow.add_node("post_editor", post_editor_agent)
workflow.add_node("publishing", publishing_agent)
workflow.add_node("rl_feedback", rl_feedback_agent)

# Define edges
workflow.add_edge("user_interaction", "brand_identity")
workflow.add_edge("brand_identity", "content_strategist")
workflow.add_edge("content_strategist", "seo")
workflow.add_edge("seo", "content_generator")
workflow.add_edge("content_generator", "post_editor")
workflow.add_edge("post_editor", "publishing")
workflow.add_edge("publishing", "rl_feedback")
workflow.add_edge("rl_feedback", END)

workflow.set_entry_point("user_interaction")
app = workflow.compile()

# Streamlit interface
st.title("BrandMate: LinkedIn Post Generator")
user_input = st.text_input("Enter your post requirements:")
if st.button("Generate Post"):
    initial_state = {"user_input": user_input}
    result = app.invoke(initial_state)
    st.write("**Generated Post:**", result["post"])
    st.write("**Metrics:**", result["metrics"])
    st.write("**Feedback:**", result["feedback"])
    st.write("**Q-table:**", result["q_table"])
```

Listing 1: LangGraph Workflow with Streamlit Deployment

This code sets up the LangGraph workflow and provides a Streamlit interface for users to input requirements and view outputs, as shown in the deployment screenshots.

4 Final Deliverables

`BrandMate` provides:

01 SEO-Optimized Post Text: Tailored LinkedIn post.

02 Recommended Hashtags: Trend-aware hashtags from `seo_agent`.

03 Schedule-to-Post: Automated scheduling recommendations.

04 Performance Tracker: Real-time metrics (likes, comments, shares).

05 Feedback Dashboard: RL-driven insights and Q-table suggestions.

Table 2: Before and After Using BrandMate

Metric	**Before**	**After**
Time to create a post	∼3–4 hours	< 15 minutes
Post performance insights	Manual brainwork	Real-time metrics + reinforcement learning
SEO optimization	Applied with third-party services / not applied	Automated, dynamic keyword and trend integration
Brand consistency	Inconsistent	Tone/style aligned with goals via strategy agents

5 Before and After Comparison

The table above shows how `BrandMate` transforms content creation, reducing post creation time, automating SEO, providing RL-driven insights, and ensuring brand consistency, enabling influencers to focus on strategy and engagement.

6 Team Contributions

The `BrandMate` project was developed by:

- **Aruni Saxena**: Designed and implemented core agents (`user_interaction_agent`, `brand_identity_agen` `content_strategist_agent`, `content_generator_agent`).
- **Anandita Saolapurkar**: Developed the `seo_agent`, `post_editor_agent`, and `publishing_agent`.
- **Rishi Barapatre**: Engineered the `rl_feedback_agent` with Q-learning.
- **Paarth Patel**: Architected the technical stack, integrating LangGraph and dependencies.

7 Additional Notes or Reflections

7.1 Development Process

Development began with user interviews, followed by a multi-agent design using LangGraph. Aruni and Anandita developed agent logic, Rishi focused on RL, and Paarth integrated the tech stack.

7.2 Challenges and Failed Attempts

- **RL Deployment Challenge**: The `rl_feedback_agent` has not been deployed live. Complex Q-table designs slowed convergence; simplified to `bold_short` and `bold_long`.
- **SEO Agent Integration**: SerpAPI rate limits required mock data fallbacks.
- **State Management**: Inconsistent `BrandMateState` updates were fixed with `TypedDict`.

7.3 Lessons Learned

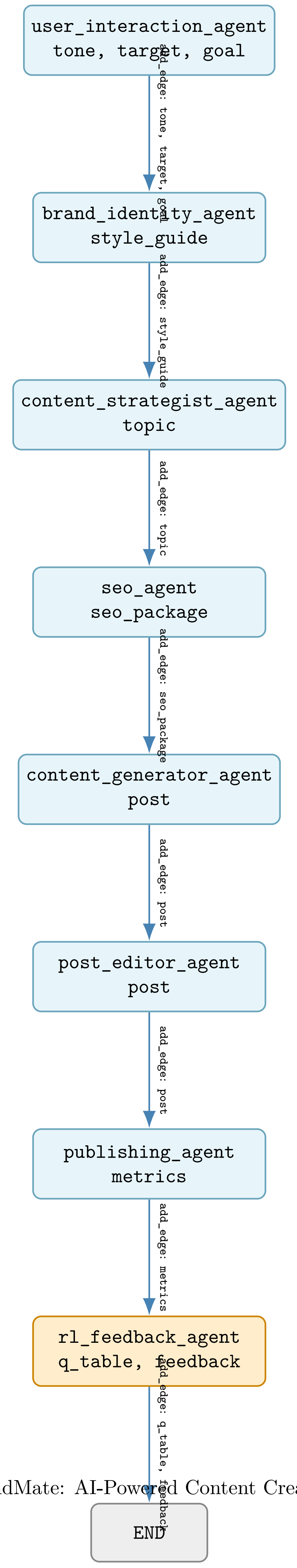
- Modular design needs robust state validation.
- RL requires careful tuning of α and γ.
- API dependencies need fallbacks.

7.4 Planned Fixes

- ▷ **RL Deployment**: Test with real LinkedIn data, expanding Q-table states.
- ▷ **Feedback Dashboard**: Develop a user-facing interface.
- ▷ **SEO Handling**: Implement caching for SerpAPI.

8 Workflow Diagram

The diagram shows the `BrandMate` workflow, with the RL agent highlighted.



BrandMate: AI-Powered Content Creation

9 Reinforcement Learning Agent

The `rl_feedback_agent` uses Q-learning to optimize post strategies.

9.1 Q-Learning Framework

Q-learning updates a Q-table mapping state-action pairs to rewards. States combine tone and post structure (e.g., `bold_short`), with actions mirroring states. The Q-table is in `state["q_table"]` (e.g., `{"bold_short": 0.0, "bold_long": 0.0}`).

The update rule is:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a')\right)$$

where:

- ▷ $Q(s,a)$: Q-value for state s and action a.
- ▷ $\alpha = 0.1$: Learning rate.
- ▷ r: Reward from metrics.
- ▷ $\gamma = 0.9$: Discount factor.
- ▷ $\max_{a'} Q(s',a')$: Maximum Q-value for the next state.

9.2 Reward Calculation

The reward is:

$$r = \frac{0.5 \cdot \text{likes} + 1.0 \cdot \text{comments} + 0.7 \cdot \text{shares}}{100}$$

Comments are prioritized (1.0), followed by shares (0.7) and likes (0.5).

9.3 Implementation

The RL agent's code is:

```python
def rl_feedback_agent(state: BrandMateState) -> BrandMateState:
    if "q_table" not in state:
        state["q_table"] = {
            "bold_short": 0.0,
            "bold_long": 0.0
        }
    current_state = f"{state['tone']}_short"
    action = current_state
    metrics = state["metrics"]
    reward = (metrics["likes"] * 0.5 + metrics["comments"] * 1.0 +
       metrics["shares"] * 0.7) / 100
    alpha, gamma = 0.1, 0.9
    state["q_table"][action] = (1 - alpha) * state["q_table"][action]
       + alpha * (reward + gamma * max(state["q_table"].values()))
    best_action = max(state["q_table"], key=state["q_table"].get)
    state["feedback"] = f"Reward: {reward:.2f}. Suggest: Use
       {best_action.replace('_', ' ')} for next post."
    print(f"Feedback: {state['feedback']}")
    print(f"Q-table: {state['q_table']}")
    return state
```

Listing 2: RL Feedback Agent

The agent:

- Initializes the Q-table.
- Forms the state (e.g., `bold_short`).
- Computes the reward from `metrics`.
- Updates the Q-table.
- Suggests the best action and updates `state["q_table"]` and `state["feedback"]`.

9.4 Addressing the Problems

The RL agent tackles:

- **Poor Performance Insight**: Provides metrics analysis and Q-table suggestions.
- **Content Evolution**: Adapts strategies via the feedback dashboard.

10 Example Execution

The workflow is invoked as:

```
initial_state = {
    "user_input": "Create a high-engagement LinkedIn post, emotional
       tone, for women startup founders, to grow followers"
}
result = app.invoke(initial_state)
print("Generated Post:", result["post"])
print("Metrics:", result["metrics"])
print("Feedback:", result["feedback"])
print("Q-table:", result["q_table"])
```

Listing 3: Invoking BrandMate

11 Project Resources and Links

The following resources document the `BrandMate` project:

- **Final Presentation Slides**: Available at `https://www.canva.com/design/DAGobAdE05U/Te5QAhm_Ymb4fe9yFNC8nA/edit`.
- **Demo Link**: The Streamlit deployment is demonstrated through three screenshots, available in the project repository and presentation, showcasing the user interface and outputs.
- **GitHub Code Repository**: Source code, including a `readme.md`, is hosted at `https://github.com/Aruni20p/BrandMate/tree/main`.
- **Video Recording**: A video of the initial deployment on Relevance AI, before transitioning to LangGraph for flexibility, is available at https://drive.google.com/file/d/1D3OgFBFEw5Fj9we0ciY0uifoU
This highlights the shift from a no-code platform to a code-based solution.

12 Conclusion

`BrandMate`, developed by Aruni Saxena, Paarth Patel, Rishi Barapatre, and Anandita Saolapurkar, addresses:

- **Consistent Brand Messaging**: Ensures style via `brand_identity_agent`.
- **Trends**: Integrates keywords via `seo_agent`.
- **Performance Insight**: RL-driven insights via `rl_feedback_agent`.
- **Burnout**: Reduces creation time to <15 minutes.

The deliverables and resources empower influencers, though RL deployment remains a challenge.