Multi-Agent AI System Simulation

```
import os
import random
import time
import matplotlib.pyplot as plt
import numpy as np import
gradio as gr
from crewai import Agent, Task, Crew, Process
import tempfile import
traceback import json
import requests from
pathlib import Path
import shutil import
logging


logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger(__name__)

class DeliveryEnvironment:     def
__init__(self, grid_size=10):
self.grid_size = grid_size
        self.city_map = self.generate_city_map()
self.delivery_requests = []
self.completed_deliveries = []
self.failed_deliveries = []
self.delivery_history = []
self.visualization_path = None

    def generate_city_map(self):
        """Generate a city map with obstacles (buildings, rivers,
etc.)"""
        city_map = np.zeros((self.grid_size, self.grid_size))


        num_buildings = int(self.grid_size * self.grid_size * 0.2)
for _ in range(num_buildings):             x, y =
random.randint(0, self.grid_size-1), random.randint(0,
self.grid_size-1)             city_map[y, x] = 2


        river_y = random.randint(2, self.grid_size-3)
for x in range(self.grid_size):             if
random.random() < 0.8:
city_map[river_y, x] = 3
```

```
        bridge_x = random.randint(1, self.grid_size-2)
city_map[river_y, bridge_x] = 0         return
city_map

    def add_delivery_request(self, source, destination, priority=1,
package_type="standard"):
        """Add a new delivery request to the system"""
request_id = len(self.delivery_requests) + 1
request = {
            "id": request_id,
"source": source,
            "destination": destination,
            "priority": priority,
            "package_type": package_type,
            "status": "pending",
            "timestamp": time.time(),
            "assigned_to": None
        }
        self.delivery_requests.append(request)
return request_id

    def get_pending_requests(self):
        """Get all pending delivery requests"""
        return [req for req in self.delivery_requests if req["status"]
== "pending"]

    def update_request_status(self, request_id, status, agent_id=None):
        """Update the status of a delivery request"""
for req in self.delivery_requests:             if
req["id"] == request_id:
req["status"] = status                 if agent_id:
req["assigned_to"] = agent_id
self.delivery_history.append({
                    "request_id": request_id,
                    "status": status,
                    "timestamp": time.time()
                })
return True         return
False

    def mark_delivery_complete(self, request_id, success=True):
        """Mark a delivery as complete or failed"""         for req
in self.delivery_requests:             if req["id"] == request_id:
req["status"] = "completed" if success else "failed"
if success:
```

```
                    self.completed_deliveries.append(req)
else:
self.failed_deliveries.append(req)                 return
True         return False

    def is_valid_position(self, position):
        """Check if a position is valid (within bounds and not an
obstacle)"""
        x, y = position         if 0 <= x < self.grid_size and
0 <= y < self.grid_size:             return self.city_map[y, x]
== 0         return False

    def visualize_delivery(self, paths=None, delivery_agents=None):
        """Visualize the city map and delivery paths"""
plt.figure(figsize=(10, 10))


        cmap = plt.cm.colors.ListedColormap(['white', 'green', 'gray',
'blue'])
        bounds = [0, 1, 2, 3, 4]
        norm = plt.cm.colors.BoundaryNorm(bounds, cmap.N)

img = plt.imshow(self.city_map, cmap=cmap, norm=norm)

        if paths:             for path, color in paths:
if path:                     x_coords, y_coords = zip(*[(p[0], p[1])
for p in path])
                    plt.plot(x_coords, y_coords, color=color,
linestyle='-', marker='o', markersize=5)

        if delivery_agents:             for agent in
delivery_agents:                 if
agent.current_position:
plt.plot(agent.current_position[0],
agent.current_position[1], 'ro', markersize=10)

        plt.title("Smart City Delivery Simulation")
plt.grid(True)

        cbar = plt.colorbar(img, ticks=[0.5, 1.5, 2.5, 3.5])
cbar.ax.set_yticklabels(['Empty Road', 'Path', 'Building',
'River'])

        plt.tight_layout()

        Path("output").mkdir(exist_ok=True)
```

```
        file_path = "output/delivery_simulation.png"
plt.savefig(file_path)
        self.visualization_path = file_path
plt.close()

        return self.visualization_path

from langchain.llms.base import LLM
from typing import Optional, List, Mapping, Any

class OpenAILLM(LLM):
    """LangChain compatible wrapper for OpenAI API."""

    api_key: str
    model_name: str = "gpt-3.5-turbo"
temperature: float = 0.7
max_tokens: int = 1000

    @property     def
_llm_type(self) -> str:
return "openai"

    def _call(self, prompt: str, stop: Optional[List[str]] = None) ->
str:         try:             headers = {
                "Authorization": f"Bearer {self.api_key}",
                "Content-Type": "application/json"
            }

            logger.debug(f"Using API key: {self.api_key[:5]}...")

            data = {
                "model": self.model_name,
                "messages": [{"role": "user", "content": prompt}],
                "temperature": self.temperature,
                "max_tokens": self.max_tokens
            }

            if stop:
data["stop"] = stop

            response = requests.post(
                "https://api.openai.com/v1/chat/completions",
headers=headers,                 data=json.dumps(data)
            )

            logger.debug(f"Response status code:
{response.status_code}")
```

```
            logger.debug(f"Response content: {response.text}")

            if response.status_code == 200:
result = response.json()
                return result["choices"][0]["message"]["content"]
else:                 logger.error(f"Error with OpenAI API:
Status
{response.status_code}")
                logger.error(response.text)
                return f"Error: API returned status code
{response.status_code}"         except Exception as e:
logger.error(f"Exception occurred: {str(e)}")
return f"Error: {str(e)}"

    @property     def _identifying_params(self) ->
Mapping[str, Any]:          return {
            "model_name": self.model_name,
            "temperature": self.temperature,
            "max_tokens": self.max_tokens
        } def

create_agents(api_key):

    llm = OpenAILLM(api_key=api_key)
planner = Agent(
        name="Chief Planning Officer (CPO)",
role="Strategic Delivery Operations Director",
goal="Optimize citywide delivery logistics with maximum
efficiency",
        backstory="""You are Prateek, a veteran logistics expert with
20 years of experience in urban delivery systems.
        After revolutionizing operations at Global Express, you were
recruited to lead this smart city's delivery network.         Your
analytical mind thrives on complex logistics challenges, and you
have a knack for anticipating bottlenecks before they occur.
You are meticulous, forward-thinking, and have an uncanny ability to
balance competing priorities.
        Your management style emphasizes data-driven decisions and
clear communication with your team.""",         verbose=True,
allow_delegation=True,         max_delegation=2,         llm=llm
    )

    navigator = Agent(
name="Master Navigator",
        role="Geospatial Intelligence and Route Optimization
```

```
Specialist",
        goal="Calculate the most efficient delivery routes while
adapting to real-time conditions",
        backstory="""You are Maya, holding a Ph.D. in computational
geography and formerly the lead algorithm designer for the world's top
navigation software.
        You've designed pathfinding systems that serve millions of
users daily, and your routing algorithms are considered industry
standards.         Having mapped over 500 cities worldwide, you
can predict traffic patterns and identify shortcuts that others
miss.
        You're fascinated by the mathematical elegance of optimal path
solutions and take pride in finding routes that save even seconds of
travel time.
        While sometimes too focused on theoretical perfection,
your routes are consistently reliable and innovative.""",
verbose=True,         llm=llm
    )


    delivery = Agent(
name="Elite Courier",
        role="Field Operations Delivery Specialist",
        goal="Execute deliveries with perfect accuracy and customer
satisfaction",
        backstory="""You are Michael Jackson, former champion cyclist
and expert urban navigator who knows every alley and shortcut in the
city.         Having completed over 10,000 successful deliveries in
your career, you hold the city record for on-time performance.
        You excel at adapting to unexpected obstacles, whether
traffic jams, weather events, or road closures.         Your customer
service skills are exceptional, making recipients feel valued with
each interaction.         While fiercely independent, you understand
the value of team coordination and real-time updates.
        You take immense pride in your perfect delivery record and
are determined to maintain it.""",         verbose=True,
llm=llm
    )

    return planner, navigator, delivery  def

create_tasks(planner, navigator, delivery):

    plan_task = Task(
        description="""As Chief Planning Officer, analyze the current
delivery request queue and optimize assignment to couriers.
```

```
        Your responsibilities:
1.   Review all pending delivery requests including source,
destination, priority level, and package type
2.   Consider current city conditions including traffic reports,
weather alerts, and courier availability
3.   Group deliveries by geographic proximity to maximize
courier efficiency
4.   Assign appropriate priority levels based on package type
and delivery urgency
5.   Create detailed delivery manifests for each courier with
clear instructions and contingency notes
6.   Monitor the overall delivery pipeline and adjust assignments as
conditions change

        Your strategic thinking should balance speed, efficiency, and
resource utilization while maintaining delivery guarantees.

        Expected format for your response:
- Summary of current delivery situation
- List of delivery assignments with rationale
- Any strategic notes for other team members
        """,
        expected_output="Comprehensive delivery assignment plan with
courier assignments and priority scheduling",         agent=planner
    )


    navigate_task = Task(
        description="""As Master Navigator, calculate optimal routes
for all assigned deliveries considering real-time city conditions.

        Your responsibilities:
1.   Analyze the city grid to identify viable pathways between
delivery points
2.   Consider known obstacles including buildings, rivers,
construction zones, and traffic congestion
3.   Calculate primary and backup routes for each delivery
4.   Estimate travel times with confidence intervals
5.   Identify optimal courier handoff points for multi-stage
deliveries
6.   Annotate routes with navigation hazards and recommended approach
vectors

        Your pathfinding algorithms should optimize for reliability
first, speed second, while avoiding known danger zones.

        Expected format for your response:
        - Route maps for each delivery (coordinate sequences)
```

```
- Estimated travel times
- Navigation hazard warnings
- Bridge and intersection strategies
        """,
        expected_output="Detailed route plans with turn-by-turn
directions and alternative paths for each delivery",
agent=navigator
    )


    deliver_task = Task(
        description="""As Elite Courier, execute package deliveries
following assigned routes while adapting to real-time conditions.

        Your responsibilities:
1.   Collect packages from designated pickup points
2.   Follow the primary navigation route while watching for
obstacles or delays
3.   Make real-time decisions about route adjustments as needed
4.   Maintain appropriate speed and safety margins during
transit
5.   Verify recipient identity and obtain delivery confirmation
6.   Report delivery status and any unusual circumstances
7.   Manage package handling according to priority and fragility
requirements

        Your field expertise and adaptive decision-making are crucial
to maintaining our perfect delivery record.

        Expected format for your response:
- Delivery execution narrative
- Status updates at key checkpoints
- Final delivery confirmation
- Any field observations relevant to future deliveries
        """,
        expected_output="Detailed delivery execution report with
status confirmations and field observations",
agent=delivery
    )

    return plan_task, navigate_task, deliver_task

def run_delivery_simulation(api_key=None, progress_callback=None):
try:         if Path("output").exists():
shutil.rmtree("output")
        Path("output").mkdir(exist_ok=True)
env = DeliveryEnvironment(grid_size=15)
```

```
        env.add_delivery_request(source=(1, 1), destination=(12, 12),
priority=4, package_type="fragile")
        env.add_delivery_request(source=(3, 7), destination=(10, 2),
priority=2, package_type="standard")
        env.add_delivery_request(source=(8, 3), destination=(2, 10),
priority=3, package_type="perishable")
        env.add_delivery_request(source=(5, 5), destination=(14, 8),
priority=5, package_type="medical")

        if progress_callback:
progress_callback("🚚 Starting Smart City Delivery
Simulation 🚚")             progress_callback(f"Current
Environment:
{env.grid_size}x{env.grid_size} city grid")
progress_callback(f"Pending Deliveries:
{len(env.get_pending_requests())}")

        if progress_callback:
progress_callback("Generating initial city map
visualization...")

        initial_map_path = env.visualize_delivery()

        if progress_callback:
progress_callback("Initial map visualization saved.")

        if not api_key:             if progress_callback:
progress_callback("No API key provided. Skipping agent simulation.")

            env.mark_delivery_complete(1, success=True)
env.mark_delivery_complete(2, success=True)
env.mark_delivery_complete(3, success=False)
env.mark_delivery_complete(4, success=True)

            sample_paths = [
                ([(1,1), (2,2), (3,3), (5,5), (8,8), (12,12)], 'red'),
                ([(3,7), (4,6), (5,5), (7,4), (10,2)], 'blue'),
                ([(8,3), (7,4), (6,5), (4,7), (2,10)], 'green'),
                ([(5,5), (7,6), (9,7), (12,8), (14,8)], 'purple')
            ]

            if progress_callback:
progress_callback("Generating final visualization with sample
paths...")

            final_map_path =
env.visualize_delivery(paths=sample_paths)
```

```
            if progress_callback:
progress_callback("\n🚚 FINAL RESULTS (SAMPLE DATA):")
progress_callback(f"Successful deliveries:
{len(env.completed_deliveries)}")
progress_callback(f"Failed deliveries:
{len(env.failed_deliveries)}")

            return {
                "initial_map": initial_map_path,
"final_map": final_map_path,
                "successful_deliveries": len(env.completed_deliveries),
                "failed_deliveries": len(env.failed_deliveries),
                "status": "Simulation completed with sample data (no
API key provided)"
            }

        if progress_callback:
progress_callback("Initializing agents with provided API key...")

        planner, navigator, delivery = create_agents(api_key)
plan_task, navigate_task, deliver_task = create_tasks(planner,
navigator, delivery)

        if progress_callback:
progress_callback("Forming delivery crew and assigning tasks...")

        delivery_crew = Crew(
            agents=[planner, navigator, delivery],
tasks=[plan_task, navigate_task, deliver_task],
verbose=True,
            process=Process.sequential
        )

        if progress_callback:             progress_callback("\n🚚
SIMULATION: Introducing a delivery edge case - package with special
handling requirements")

        env.add_delivery_request(source=(2, 2), destination=(13, 13),
priority=5, package_type="fragile-SPECIAL-HANDLING")

        if progress_callback:             progress_callback("\n🚚
RUNNING AGENT SIMULATION: Agent interactions beginning...")

        try:             result =
delivery_crew.kickoff()             if
progress_callback:
```

```
                progress_callback("🚚 Agent simulation
completed successfully")         except Exception as e:
if progress_callback:                 progress_callback(f"⚠
Error in agent simulation:
{str(e)}")
                progress_callback("Continuing with visualization...")
result = "Simulation encountered an error but will continue with
visualization"         for i, req in
enumerate(env.delivery_requests):

            success = random.random() > 0.2
            env.mark_delivery_complete(req["id"], success=success)

        sample_paths = [
            ([(1,1), (2,2), (3,3), (5,5), (8,8), (12,12)], 'red'),
            ([(3,7), (4,6), (5,5), (7,4), (10,2)], 'blue'),
            ([(8,3), (7,4), (6,5), (4,7), (2,10)], 'green'),
            ([(5,5), (7,6), (9,7), (12,8), (14,8)], 'purple'),
            ([(2,2), (3,3), (5,6), (8,9), (11,11), (13,13)], 'orange')
        ]

        if progress_callback:             progress_callback("\n
GENERATING FINAL VISUALIZATION: Creating delivery paths map...")

        final_map_path = env.visualize_delivery(paths=sample_paths)


        if progress_callback:
progress_callback("\n🚚 FINAL RESULTS:")
progress_callback(f"Total requests processed:
{len(env.delivery_requests)}")
progress_callback(f"Successful deliveries:
{len(env.completed_deliveries)}
({len(env.completed_deliveries)/len(env.delivery_requests)*100:.1f}
%)")             progress_callback(f"Failed
deliveries:
{len(env.failed_deliveries)}
({len(env.failed_deliveries)/len(env.delivery_requests)*100:.1f}%)")

        return {
            "initial_map": initial_map_path,
            "final_map": final_map_path,
            "successful_deliveries": len(env.completed_deliveries),
            "failed_deliveries": len(env.failed_deliveries),
            "status": "Simulation completed successfully with agent
interaction"
        }
```

```
    except Exception as e:         error_message = f"Error
during simulation: {str(e)}\ n{traceback.format_exc()}"
if progress_callback:
progress_callback(error_message)         return {
            "status": "Error",
            "error_message": error_message
        }

def create_gradio_interface():     with gr.Blocks(title="Smart
City Delivery Simulation") as app:         gr.Markdown("# 🚚
Smart City Delivery Simulation")
        gr.Markdown("Simulate a multi-agent delivery system with
AIpowered planning, navigation, and execution.")

        with gr.Row():
api_key_input = gr.Textbox(
label="OpenAI API Key",
                placeholder="Enter your OpenAI API key here...",
type="password"
            )

        with gr.Row():             run_button = gr.Button("🚚
Run Simulation", variant="primary")             demo_button =
gr.Button("🚚 Demo Mode (No API Key)")

        with gr.Row():
progress_output = gr.Textbox(
label="Simulation Progress",
                placeholder="Simulation progress will appear here...",
lines=10
            )

        with gr.Row():
with gr.Column():
                initial_map = gr.Image(label="Initial City Map")
with gr.Column():                 final_map = gr.Image(label="Final
Delivery Routes")

        with gr.Row():             with gr.Column():
successful_deliveries = gr.Number(label="Successful Deliveries",
value=0)             with gr.Column():
failed_deliveries = gr.Number(label="Failed Deliveries", value=0)
with gr.Column():
```

```
                success_rate = gr.Number(label="Success Rate (%)",
value=0)

        def update_progress(message):
            return message

        def run_simulation(api_key):
progress_list = []

            def progress_callback(message):
progress_list.append(message)
return "\n".join(progress_list)

            results = run_delivery_simulation(api_key,
progress_callback)

            total_deliveries = results["successful_deliveries"] +
results["failed_deliveries"]
            success_rate_value = (results["successful_deliveries"] /
total_deliveries * 100) if total_deliveries > 0 else 0

            return {
                progress_output: "\n".join(progress_list),
initial_map: results.get("initial_map"),
final_map: results.get("final_map"),
successful_deliveries: results["successful_deliveries"],
                failed_deliveries: results["failed_deliveries"],
success_rate: success_rate_value             }

        def run_demo():
progress_list = []

            def progress_callback(message):
progress_list.append(message)
return "\n".join(progress_list)

            results = run_delivery_simulation(None, progress_callback)

            total_deliveries = results["successful_deliveries"] +
results["failed_deliveries"]
            success_rate_value = (results["successful_deliveries"] /
total_deliveries * 100) if total_deliveries > 0 else 0

            return {
                progress_output: "\n".join(progress_list),
initial_map: results.get("initial_map"),
final_map: results.get("final_map"),
successful_deliveries:
```

```
results["successful_deliveries"],
                failed_deliveries: results["failed_deliveries"],
success_rate: success_rate_value             }

        run_button.click(
fn=run_simulation,
inputs=[api_key_input],
            outputs=[progress_output, initial_map, final_map,
successful_deliveries, failed_deliveries, success_rate]
)

        demo_button.click(
fn=run_demo,
inputs=[],
            outputs=[progress_output, initial_map, final_map,
successful_deliveries, failed_deliveries, success_rate]
)     return app

if __name__ == "__main__":     app
= create_gradio_interface()
app.launch()
```

```
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio app to work, sharing must be enabled. Automatically setting `share=True` (you can turn this off by setting `share=False` in `launch()` explicitly).

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://319ba04e40d9919dcf.gradio.live
This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (https://huggingface.co/spaces)


<IPython.core.display.HTML object>
```

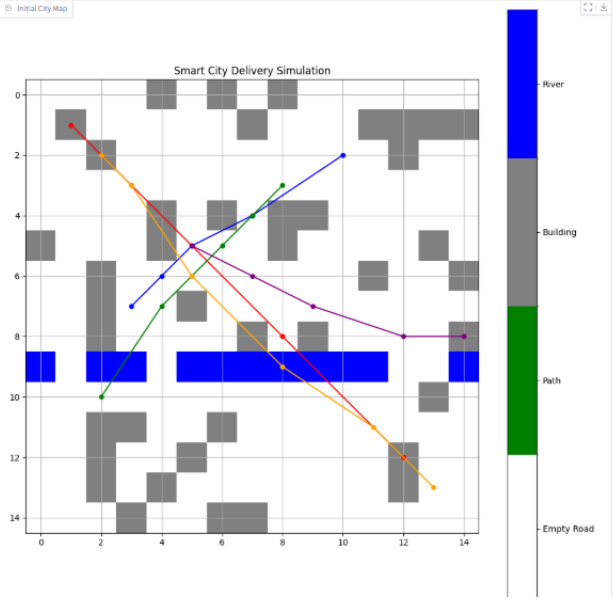
Output:

Run Simulation | Demo Mode (No API Key)

Simulation Progress

```
🚚 Starting Smart City Delivery Simulation 🚚
Current Environment: 15x15 city grid
Pending Deliveries: 4
Generating initial city map visualization...
Initial map visualization saved.
Initializing agents with provided API key...
Forming delivery crew and assigning tasks...

🔺 SIMULATION: Introducing a delivery edge case - package with special handling requirements

🔄 RUNNING AGENT SIMULATION: Agent interactions beginning...
⚠️ Error in agent simulation: litellm.RateLimitError: RateLimitError: OpenAIException - You exceeded your current quota, please check your plan and billing details. For more information on this error, read the docs: https://platform.openai.com/docs/guides/error-codes/api-errors.
Continuing with visualization...

🗺️ GENERATING FINAL VISUALIZATION: Creating delivery paths map...

📊 FINAL RESULTS:
Total requests processed: 5
Successful deliveries: 4 (80.0%)
Failed deliveries: 1 (20.0%)
```

Initial City Map



Final Delivery Routes