# ДОДАТОК А

## Модуль ініціалізації логістичного графа

```
      import networkx as nx
import matplotlib.pyplot as plt


def create_ukraine_logistics_graph():
    """
    Створює зважений граф ключових логістичних маршрутів України.
    Кожне ребро містить базові атрибути: distance (відстань) та base_risk
(базовий ризик).
    """
    G = nx.Graph()

    # Додавання ключових логістичних хабів (міст)
    cities = [
        "Київ", "Житомир", "Рівне", "Львів",
        "Вінниця", "Хмельницький", "Тернопіль",
        "Одеса", "Умань", "Дніпро", "Полтава", "Харків"
    ]
    G.add_nodes_from(cities)

    # Визначення маршрутів (ребер): (Місто_А, Місто_Б, {атрибути})
    # Відстані є приблизними для цілей симуляції
    edges = [
        ("Київ", "Житомир", {'distance': 140, 'base_risk': 0.1}),
        ("Житомир", "Рівне", {'distance': 190, 'base_risk': 0.1}),
        ("Рівне", "Львів", {'distance': 210, 'base_risk': 0.2}),

        ("Київ", "Полтава", {'distance': 340, 'base_risk': 0.1}),
        ("Полтава", "Харків", {'distance': 140, 'base_risk': 0.3}),
        ("Полтава", "Дніпро", {'distance': 200, 'base_risk': 0.2}),
        ("Київ", "Дніпро", {'distance': 450, 'base_risk': 0.3}),

        ("Київ", "Умань", {'distance': 210, 'base_risk': 0.1}),
        ("Умань", "Одеса", {'distance': 270, 'base_risk': 0.1}),

        ("Київ", "Вінниця", {'distance': 260, 'base_risk': 0.1}),
        ("Умань", "Вінниця", {'distance': 160, 'base_risk': 0.2}),
        ("Вінниця", "Хмельницький", {'distance': 120, 'base_risk': 0.1}),
```

```
        ("Хмельницький", "Тернопіль", {'distance': 110, 'base_risk': 0.2}),
        ("Тернопіль", "Львів", {'distance': 130, 'base_risk': 0.1}),
    ]

    G.add_edges_from(edges)
    return G


def draw_graph(G, current_node=None, problem_edges=None, path=None):
    """
    Функція для візуалізації графа за допомогою Matplotlib.
    """
    plt.figure(figsize=(12, 8))

    # Приблизне географічне розташування міст для красивого виводу
    pos = {
        "Львів": (0, 3), "Рівне": (1, 4), "Тернопіль": (1, 2.5),
        "Житомир": (2.5, 4), "Хмельницький": (2, 2.5), "Вінниця": (3, 2),
        "Київ": (4, 4), "Умань": (4, 1.5), "Одеса": (4, 0),
        "Полтава": (6, 3.5), "Дніпро": (6, 1.5), "Харків": (8, 4)
    }

    # Малюємо всі вузли та стандартні ребра (сірі)
    nx.draw_networkx_nodes(G, pos, node_color='lightblue', node_size=2000)
    nx.draw_networkx_edges(G, pos, edge_color='gray', width=2)
    nx.draw_networkx_labels(G, pos, font_size=10, font_family="sans-serif",
font_weight="bold")

    # Підсвічуємо проблемні ділянки (червоні), якщо вони є
    if problem_edges:
        nx.draw_networkx_edges(G, pos, edgelist=problem_edges,
edge_color='red', width=4)

    # Підсвічуємо поточний маршрут (зелений)
    if path:
        path_edges = [(path[i], path[i + 1]) for i in range(len(path) - 1)]
        nx.draw_networkx_edges(G, pos, edgelist=path_edges,
edge_color='green', width=4)

    # Підсвічуємо поточне положення авто (жовтий)
    if current_node:
        nx.draw_networkx_nodes(G, pos, nodelist=[current_node],
node_color='yellow', node_size=2500)
```

```
    # Виводимо відстані на ребрах
    edge_labels = nx.get_edge_attributes(G, 'distance')
    nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels,
font_size=8)

    plt.title("Інтелектуальна система маршрутизації логістичної мережі
України", fontsize=14)
    plt.axis('off')
    plt.show()


# Блок тестування генерації графа
if __name__ == "__main__":
    logistics_graph = create_ukraine_logistics_graph()
    draw_graph(logistics_graph)
```

# ДОДАТОК Б

## Модуль емуляції інфраструктурного середовища та функціонування Edge AI

```
      import random


class InfrastructureEnvironment:
    """
    Клас для імітації динамічних змін у логістичному графі.
    Випадковим чином генерує затори та перекриття доріг.
    """

    def __init__(self, graph):
        self.graph = graph

    def generate_dynamic_constraints(self):
        problem_edges = []
        for u, v, data in self.graph.edges(data=True):
            # Імітація ймовірності виникнення інфраструктурної проблеми
(15%)
            if random.random() < 0.15:
                data['current_risk'] = data['base_risk'] +
random.uniform(0.6, 0.9)
                problem_edges.append((u, v))
            else:
                data['current_risk'] = data['base_risk'] +
random.uniform(0.0, 0.1)
        return problem_edges


class EdgeAIEmulator:
    """
    Емулятор бортового модуля штучного інтелекту.
    Аналізує локальні дані та виявляє перешкоди на маршруті.
    """

    def __init__(self, current_node, risk_threshold=0.7):
        self.current_node = current_node
        self.risk_threshold = risk_threshold
```

```
    def scan_ahead(self, graph, next_node):
        edge_data = graph.get_edge_data(self.current_node, next_node)
        if not edge_data:
            return False

        # Отримання прогнозованого ризику з урахуванням динамічних обмежень
        predicted_risk = edge_data.get('current_risk',
edge_data['base_risk'])

        # Якщо ризик вищий за поріг, приймається рішення про блокування
шляху
        return predicted_risk > self.risk_threshold
```

# ДОДАТОК В

## Модуль алгоритму адаптивної маршрутизації

```
import networkx as nx


class AdaptiveRouter:
    """
    Клас для динамічного прокладання маршруту в умовах інфраструктурних обмежень.
    Взаємодіє з Edge AI емулятором для обходу перешкод у реальному часі.
    """

    def __init__(self, graph, start_node, target_node, emulator):
        self.graph = graph
        self.current_node = start_node
        self.target_node = target_node
        self.emulator = emulator
        self.path_taken = [start_node]

    def calculate_optimal_path(self):
        try:
            # Використовуємо алгоритм пошуку найкоротшого шляху з урахуванням ризику
            path = nx.shortest_path(
                self.graph,
                source=self.current_node,
                target=self.target_node,
                weight='current_risk'
            )
            return path
        except nx.NetworkXNoPath:
            return None

    def run_simulation(self):
        print(f"--- Початок маршруту: {self.current_node} -> {self.target_node} ---")

        while self.current_node != self.target_node:
            planned_path = self.calculate_optimal_path()
```

```
            if not planned_path:
                print("[КРИТИЧНО] Немає доступного шляху до цілі. Мережа
заблокована.")
                break

            next_node = planned_path[1]

            # Локальне сканування інфраструктури модулем Edge AI
            if self.emulator.scan_ahead(self.graph, next_node):
                print(f"[УВАГА] Edge AI виявив критичну перешкоду на шляху
до міста {next_node}!")
                print("Ініційовано перерахунок маршруту...")

                # Штучне блокування ребра для подальшого обходу
                self.graph[self.current_node][next_node]['current_risk'] =
float('inf')
                continue

            # Успішне проходження ділянки
            self.current_node = next_node
            self.emulator.current_node = next_node
            self.path_taken.append(self.current_node)
            print(f"Рух підтверджено. Транспортний засіб прибув у:
{self.current_node}")

        print(f"--- Ціль досягнуто! Фактичний пройдений маршрут: {' ->
'.join(self.path_taken)} ---")
        return self.path_taken
```

# ДОДАТОК Г

## Головний модуль симуляції та тестування системи

```
      from logistics_graph import create_ukraine_logistics_graph,
draw_graph

from edge_emulator import InfrastructureEnvironment, EdgeAIEmulator

from adaptive_router import AdaptiveRouter


def main():
    print("--- Ініціалізація системи адаптивного управління логістикою ---
")

    # 1. Створюємо базовий граф транспортної мережі
    graph = create_ukraine_logistics_graph()

    # Налаштовуємо параметри поїздки
    start_city = "Київ"
    target_city = "Львів"

    # 2. Генеруємо динамічні інфраструктурні обмеження (затори/аварії)
    environment = InfrastructureEnvironment(graph)
    problem_edges = environment.generate_dynamic_constraints()

    print("\n[ІНФРАСТРУКТУРА] Згенеровані проблемні ділянки з високим
ризиком:")
    if problem_edges:
        for u, v in problem_edges:
            print(f"  - Перекриття/Затор на ділянці: {u} <-> {v}")
    else:
        print("  - Проблемних ділянок не виявлено. Шлях вільний.")
    print("-" * 60)

    # 3. Ініціалізуємо бортовий Edge AI модуль
    edge_emulator = EdgeAIEmulator(current_node=start_city,
risk_threshold=0.6)

    # 4. Запускаємо адаптивний маршрутизатор
    router = AdaptiveRouter(graph, start_city, target_city, edge_emulator)
```

```
    actual_path = router.run_simulation()

    # 5. Виводимо графічний результат
    if actual_path:
        print("\n[СИСТЕМА] Запуск модуля візуалізації...")
        draw_graph(graph, current_node=target_city,
problem_edges=problem_edges, path=actual_path)
    else:
        print("\n[СИСТЕМА] Неможливо побудувати маршрут. Візуалізація
скасована.")


if __name__ == "__main__":
    main()
```

# ДОДАТОК Д

## Модуль хмарної панелі диспетчеризації

```
      import dash
from dash import dcc, html
import plotly.express as px
import pandas as pd

app = dash.Dash(__name__)
#python dispatcher_app.py
# Імітація даних від різних вантажівок
df = pd.DataFrame({
    'Вантажівка': ['Truck 01 (Київ-Львів)', 'Truck 02 (Одеса-Умань)',
'Truck 03 (Дніпро-Полтава)'],
    'lat': [50.5, 47.5, 48.9],
    'lon': [26.5, 30.2, 34.8],
    'Статус': ['Edge-only (Автономний)', 'Cloud-sync (Синхронізовано)',
'Hybrid (Нестабільний)'],
    'Колір': ['red', 'green', 'orange'],
    'Розмір': [15, 10, 10]
})

# Створення географічної карти України (оновлено для нових версій Plotly)
fig = px.scatter_map(
    df, lat="lat", lon="lon", hover_name="Вантажівка",
    hover_data=["Статус"], color="Колір", size="Розмір",
    color_discrete_map={"red": "red", "green": "green", "orange":
"orange"},
    zoom=5, height=600, title="Інтерактивна карта транспортної мережі"
)
fig.update_layout(map_style="carto-positron", showlegend=False)

# Верстка інтерфейсу
app.layout = html.Div(style={'fontFamily': 'Arial, sans-serif', 'padding':
'20px'}, children=[
    html.H1("🌐 Глобальна панель диспетчеризації (Cloud Layer)"),
    html.Hr(),

    html.Div(style={'display': 'flex', 'flexDirection': 'row'}, children=[
        # Ліва панель з алертами
        html.Div(style={'width': '30%', 'paddingRight': '20px'}, children=[
```

```
            html.H3("Статус логістичної мережі"),
            html.Div(
                style={'backgroundColor': '#e8f5e9', 'padding': '10px',
'marginBottom': '10px', 'borderRadius': '5px'},
                children=[
                    html.B("🟢 Truck 02: "), "Зв'язок стабільний. Маршрут
згідно з планом."
                ]),
            html.Div(
                style={'backgroundColor': '#fff3e0', 'padding': '10px',
'marginBottom': '10px', 'borderRadius': '5px'},
                children=[
                    html.B("🟡 Truck 03: "), "Часткова втрата пакетів
телеметрії. Гібридний режим."
                ]),
            html.Div(
                style={'backgroundColor': '#ffebee', 'padding': '10px',
'marginBottom': '10px', 'borderRadius': '5px',
                       'borderLeft': '5px solid red'}, children=[
                    html.B("🔴 Truck 01: "),
                    "АВТОНОМНИЙ РЕЖИМ. Інфраструктурна перешкода! Маршрут
адаптовано модулем Edge AI."
                ]),
        ]),

        # Права панель з картою
        html.Div(style={'width': '70%'}, children=[
            dcc.Graph(figure=fig)
        ])
    ])
])

if __name__ == '__main__':
    # Оновлена команда запуску для нових версій Dash
    app.run(debug=True)
```

# ДОДАТОК Е

## Модуль інтерфейсу бортового термінала водія

```
      import streamlit as st
import pandas as pd

#streamlit run driver_app.py

# Налаштування сторінки
st.set_page_config(layout="wide", page_title="Edge AI Бортовий термінал")

st.title("🚛 Бортовий термінал адаптивної логістики (Edge AI)")
st.markdown("---")

# Бічна панель з телеметрією
st.sidebar.header("📡 Телеметрія ТЗ")
st.sidebar.metric(label="Швидкість", value="75 км/год", delta="-5 км/год")
st.sidebar.metric(label="Рівень пального", value="42%", delta="-2%")
st.sidebar.metric(label="ETA до Львова", value="3 год 15 хв",
delta="Оновлено")
st.sidebar.info("Режим обчислень: **Edge-only** (Автономний)")

# Основна панель: Імітація роботи Edge AI
st.error("⚠️ **[КРИТИЧНО]** Edge AI виявив інфраструктурну перешкоду
(затор/аварія) на трасі Рівне - Львів.")
st.success("✅ **[АДАПТАЦІЯ]** Маршрут успішно перераховано в обхід
перешкоди. Рух продовжується.")

st.subheader("📍 Локальна карта навігації (Поточний відрізок)")

# Координати для імітації маршруту (Київ -> Житомир -> Тернопіль -> Львів)
route_data = pd.DataFrame({
    'lat': [50.4501, 50.2547, 49.5535, 49.8397],
    'lon': [30.5234, 28.6586, 25.5948, 24.0297]
})
# Рендеринг карти
st.map(route_data, zoom=6)

st.caption("Система працює в автономному режимі. Зв'язок із диспетчерською
тимчасово відсутній.")
```

# ДОДАТОК Ж

Відомість кваліфікаційної роботи

| Позначення | Найменування | Дод. відомості |
|---|---|---|
| | Текстові документи | |
| **1.** | Пояснювальна записка | 79 с. |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | Інші документи | |
| | | |
| **2.** | Презентаційні матеріали | __ плакатів |
| | | |
| | | |
| | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| . | . | Прізвище та ініціали. | Підп. | Дата | | | |
| Розробив | | Георгієва Я. В. | | | Інтелектуальна система синтезу персональних освітніх планів засобами семантичного мапінгу, трансформерних NLP-моделей та онтологічного інжинірингу | Шифр групи | Код напр./спец. |
| Перевірив | | Гриньова О.Є. | | | | ІТШІ-22-1 | 122 |
| Н.контр. | | Малєєва І.А. | | | | ХНУРЕ кафедра ШІ | |
| Затв. | | Чала Л.Е. | | | | | |