ДОДАТОК А

Фрагменти програмної реалізації нейромережевого агента

Лістинг А.1 – Фрагменти програмної реалізації нейромережевого агента та простору дій світлофорного керування з файлу actions.py

```
class PhaseSelectionActionSpace(BaseActionSpace):
    def apply(self, actions):
        for ts_idx, ts in enumerate(self.traffic_signals.values()):
            ts.action_to_phase(actions[ts_idx])

    def get_size(self):
        return max([ts.controller.n_phases for ts in self.traffic_signals.
values()])

    def get_mask(self):
        masks = []
        for ts in self.traffic_signals.values():
            mask = ts.controller.get_allowable_phase_switches()
            mask = pad_list(mask, self.get_size())
            masks.append(mask)
        return masks


class PhaseSwitchActionSpace(BaseActionSpace):
    def apply(self, actions):
        self._check_actions_type(actions)
        for ts_idx, ts in enumerate(self.traffic_signals.values()):
            current_phase = ts.controller.program.current_phase_index
            if actions[ts_idx] == 1:
                next_phase = (current_phase + 1)% ts.controller.n_phases
                ts.action_to_phase(next_phase)
            else:
                ts.action_to_phase(current_phase)

    def get_size(self):
        return 2

    def get_mask(self):
        masks = []
        for ts in self.traffic_signals.values():
            allowable_switches = ts.controller.get_allowable_phase_switche
s()
            n_phases = ts.controller.n_phases
            current_phase_index = ts.controller.program.current_phase_inde
x
            next_phase_index = (current_phase_index + 1)% n_phases
            mask = [0, 0]
            if allowable_switches[current_phase_index]:
                mask[0] = 1
            if allowable_switches[next_phase_index]:
                mask[1] = 1
```

```
            masks.append(mask)
        return masks
```

Лістинг А.2 – Фрагмент формування стану агента з файлу observations.py

```
class PositionMatrix(BaseObservationSpace):
    max_lanes_per_direction = 6
    max_lane_speed = 15.0
    max_lane_length = 500
    max_phases = 20
    max_n_controlled_lanes = 16

    def __init__(self, config, parsed_network, traffic_signals, simulator_
backend):
        super(PositionMatrix, self).__init__(
            config, parsed_network, traffic_signals, simulator_backend
        )
        self.visibility = config.signal["visibility"]
        self.lane_features = self._get_lane_features()
        self.obs_noise_std = config.signal.get("obs_noise_std", 0.00)

    def get_state(self):
        states = []
        for ts in self.traffic_signals.values():
            state = []
            for lane in ts.incoming_lanes:
                state.extend(self.lane_features[lane])
                lane_results = ts.sub_results["lane"][lane]
                n_queued = lane_results["n_queued"]
                occupancy = lane_results["occupancy"]
                mean_speed = lane_results["mean_speed"]
                state.extend([n_queued, occupancy, mean_speed])
            state = pad_list(state, self.get_state_size() - self.max_phase
s, -1)
            phase_id = pad_list(ts.phase_id, self.max_phases)
            state.extend(phase_id)
            states.append(state)
        return states

    def get_state_size(self):
        return self.max_n_controlled_lanes * (3 + 9) + self.max_phases


class LaneFeatures(BaseObservationSpace):
    max_lanes_per_direction = 6
    max_lane_speed = 15.0
    max_lane_length = 500
    max_phases = 20
    max_n_controlled_lanes = 16

    def __init__(self, config, parsed_network, traffic_signals, simulator_
```

```
backend):
        super(LaneFeatures, self).__init__(
            config, parsed_network, traffic_signals, simulator_backend
        )
        self.visibility = config.signal["visibility"]
        self.dropout_prob = config.signal.get("obs_dropout_prob", 0.0)
        self.lane_features = self._get_lane_features()
        self.reset_dropped_lanes()
```

Лістинг А.3 – Фрагмент оновлення статистики світлофорного об’єкта з файлу traffic_signal.py

```
class TrafficSignal(BaseTrafficSignal):
    def __init__(self, id, config, simulator):
        super(TrafficSignal, self).__init__(id, config, simulator)
        self.controller = TSController(id, config, simulator)
        self.incoming_lanes = config["incoming_lanes"]
        self.outgoing_lanes = config["outgoing_lanes"]

    def update_stats(self, sub_results):
        self.sub_results = sub_results
        self.n_queued = 0
        self.occupancy = 0
        self.mean_speed = 0
        self.mean_delay = 0
        self.average_wait_time = 0
        self.average_travel_time = 0
        self.inc_position_matrices = {}
        max_speeds = self.simulator.parsed_network.lane_max_speeds
        for lane in self.incoming_lanes:
            lane_results = sub_results["lane"][lane]
            self.n_queued += lane_results["n_queued"]
            self.occupancy += lane_results["occupancy"]
            self.mean_speed += lane_results["mean_speed"]
            self.mean_delay += 1 - lane_results["mean_speed"] / max_speeds
[lane]
            self.average_travel_time += lane_results["average_travel_time"
]
            self.average_wait_time += lane_results["average_wait_time"]
            pos_mat = sub_results["lane"][lane]["position_matrix"]
            self.inc_position_matrices[lane] = pos_mat[-self.config["visib
ility"] :]

        self.occupancy /= len(self.incoming_lanes)
        self.mean_speed /= len(self.incoming_lanes)
        self.mean_delay /= len(self.incoming_lanes)
        self.average_travel_time /= len(self.incoming_lanes)
        self.average_wait_time /= len(self.incoming_lanes)
        self.outgoing_occupancy = 0
        self.out_position_matrices = {}
        for lane in self.outgoing_lanes:
            lane_results = sub_results["lane"][lane]
            self.outgoing_occupancy += lane_results["occupancy"]
            pos_mat = sub_results["lane"][lane]["position_matrix"]
```

```
            self.out_position_matrices[lane] = pos_mat[: self.config["visi
bility"]]
        self.outgoing_occupancy /= len(self.outgoing_lanes)

        self.time_on_phase = self.controller.norm_time_on_phase
        self.phase_id = self.controller.phase_one_hot
        self.pressure = np.abs(self.occupancy - self.outgoing_occupancy).i
tem()
        self.sim_step = self.simulator.sim_step / 3600
```

Лістинг А.4 – Фрагмент функцій винагороди з файлу reward.py

```
class QueueLength(BaseRewardFunction):
    def get_global_reward(self):
        fc = self.config.misc["flickering_coef"]
        reward = 1e-6
        reward += fc * self.metrics.flickering_signal
        reward += self.metrics.n_queued
        return -1 * reward

    def get_local_reward(self):
        fc = self.config.misc["flickering_coef"]
        gamma = self.config.misc["reward_gamma"]
        k_hop_neighbors = self.parsed_network.k_hop_neighbors
        local_rewards = {
            ts_id: -fc * ts.controller.program.phase_changed - ts.n_queued
- 1e-6
            for ts_id, ts in self.traffic_signals.items()
        }
        rewards = {}
        for ts_id in self.traffic_signals.keys():
            rewards[ts_id] = local_rewards[ts_id]
            for k in range(1, len(self.traffic_signals.keys())):
                neighbors_k = k_hop_neighbors[ts_id].get(k, [])
                for neighbor_ts_id in neighbors_k:
                    rewards[ts_id] += gammak * local_rewards[neighbor_ts_i
d]
        return list(rewards.values())


class MaxPressure(BaseRewardFunction):
    def get_local_reward(self):
        fc = self.config.misc["flickering_coef"]
        gamma = self.config.misc["reward_gamma"]
        k_hop_neighbors = self.parsed_network.k_hop_neighbors
        local_rewards = {
            ts_id: -fc * ts.controller.program.phase_changed - ts.pressure
- 1e-6
            for ts_id, ts in self.traffic_signals.items()
        }
        rewards = {}
        for ts_id in self.traffic_signals.keys():
            rewards[ts_id] = local_rewards[ts_id]
            for k in range(1, len(self.traffic_signals.keys())):
                neighbors_k = k_hop_neighbors[ts_id].get(k, [])
```

```
                for neighbor_ts_id in neighbors_k:
                    rewards[ts_id] += gammak * local_rewards[neighbor_ts_i
d]
        return list(rewards.values())
```

Лістинг А.5 – Фрагмент нейромережевого агента з файлу rl_controller.py

```
class LaneAttentionAggregator(nn.Module):
    def __init__(
        self,
        static_feat_dim=9,
        dynamic_feat_dim=10,
        phase_id_dim=20,
        hidden_dim=64,
        n_heads=4,
        device="cpu",
    ):
        super(LaneAttentionAggregator, self).__init__()
        self.device = device
        self.hidden_dim = hidden_dim
        self.k = nn.Linear(static_feat_dim + dynamic_feat_dim, hidden_dim)
        self.v = nn.Linear(static_feat_dim + dynamic_feat_dim, hidden_dim)
        self.q = nn.Linear(phase_id_dim, hidden_dim)
        self.attn = nn.MultiheadAttention(
            embed_dim=hidden_dim, num_heads=n_heads, batch_first=True
        )

    def forward(self, static_feats, dynamic_feats, phase_ids):
        lane_feats = torch.cat((static_feats, dynamic_feats), dim=-1)
        k, q, v = self.k(lane_feats), self.q(phase_ids), self.v(lane_feats
)
        x, _ = self.attn(q, k, v, need_weights=False)
        x = x.squeeze(1)
        return x


class GraphAttentionComm(nn.Module):
    def __init__(self, adjacency_matrix, hidden_dim=64, n_heads=4):
        super(GraphAttentionComm, self).__init__()
        self.n_heads = n_heads
        self.hidden_dim = hidden_dim
        self.adjacency_matrix = adjacency_matrix
        self.comm = nn.MultiheadAttention(
            embed_dim=hidden_dim, num_heads=n_heads, batch_first=True
        )

    def forward(self, x, dropout=0.0):
        x_dropped = F.dropout(x, p=dropout, training=True) if dropout > 0
else x
        attn_mask = (1 - self.adjacency_matrix).clone()
        num_agents = attn_mask.shape[0]
        for i in range(num_agents):
            attn_mask[i, i] = 0
```

```
        attn_mask = attn_mask.to(torch.bool)
        attn_out, _ = self.comm(
            x_dropped, x_dropped, x_dropped, attn_mask=attn_mask, need_wei
ghts=False
        )
        out = x + attn_out
        return out


class TSCAgent(nn.Module):
    def __init__(self, n_agents, n_actions, hidden_dim, max_n_lanes):
        super(TSCAgent, self).__init__()
        self.n_agents = n_agents
        self.n_actions = n_actions
        self.hidden_dim = hidden_dim
        self.max_n_lanes = max_n_lanes
        self.lane_obs_encoder = LaneAttentionAggregator(hidden_dim=hidden_
dim)
        self.rnn = nn.GRUCell(hidden_dim, hidden_dim)
        self.fc_out = nn.Linear(hidden_dim, n_actions)

    def forward(self, static_lane_feats, pos_mats, phase_ids, hidden_state
, kwargs):
        x = self.lane_obs_encoder(static_lane_feats, pos_mats, phase_ids)
        h_in = hidden_state.reshape(-1, self.hidden_dim)
        h = self.rnn(x, h_in)
        q = self.fc_out(h)
        return q, h


class TSCGraphAgent(nn.Module):
    def __init__(
        self,
        n_agents,
        n_actions,
        hidden_dim,
        max_n_lanes,
        adjacency_matrix,
        graph,
        n_heads=4,
    ):
        super(TSCGraphAgent, self).__init__()
        self.graph = graph
        self.n_agents = n_agents
        self.n_actions = n_actions
        self.hidden_dim = hidden_dim
        self.max_n_lanes = max_n_lanes
        self.lane_obs_encoder = LaneAttentionAggregator(hidden_dim=hidden_
dim)
        adj = self._get_adjcency_matrix(adjacency_matrix)
        self.communicate = GraphAttentionComm(adj, hidden_dim, n_heads)
        self.rnn = nn.GRUCell(hidden_dim, hidden_dim)
        self.fc_out = nn.Linear(hidden_dim, n_actions)

    def forward(
        self, static_lane_feats, pos_mats, phase_ids, hidden_state, dropou
```

```
t=0.0
    ):
        x = self.lane_obs_encoder(static_lane_feats, pos_mats, phase_ids)
        bs = x.shape[0] // self.n_agents
        h_in = hidden_state.reshape(-1, self.hidden_dim)
        h = self.rnn(x, h_in)
        h_ = h.view(bs, self.n_agents, -1)
        h_ = self.communicate(h_, dropout=dropout)
        h_ = h_.view(bs * self.n_agents, -1)
        q = self.fc_out(h_)
        return q, h
```

Лістинг А.6 – Фрагменти збору метрик і визначення сусідніх агентів з файлів retriever.py, metrics.py і network_parser.py

```
      class Retriever(BaseRetriever):
    tc = {
        "n_vehicles": tc.LAST_STEP_VEHICLE_NUMBER,
        "n_queued": tc.LAST_STEP_VEHICLE_HALTING_NUMBER,
        "mean_speed": tc.LAST_STEP_MEAN_SPEED,
        "occupancy": tc.LAST_STEP_OCCUPANCY,
        "wait_time": tc.VAR_WAITING_TIME,
        "travel_time": tc.VAR_CURRENT_TRAVELTIME,
        "phase": tc.TL_CURRENT_PHASE,
        "vehicle_id": tc.LAST_STEP_VEHICLE_ID_LIST,
    }

    def _subscribe_to_lane_vars(self):
        for lane in self.parsed_network.lanes:
            self.traci.lane.subscribe(
                lane,
                [
                    self.tc["n_vehicles"],
                    self.tc["n_queued"],
                    self.tc["mean_speed"],
                    self.tc["occupancy"],
                    self.tc["wait_time"],
                    self.tc["travel_time"],
                    self.tc["vehicle_id"],
                ],
            )


class MetricsParser(BaseMetricsParser):
    @property
    def n_vehicles(self):
        lane_measurements = self.simulator.step_measurements["lane"]
        return sum(data["n_vehicles"] for data in
lane_measurements.values())

    @property
    def n_queued(self):
        lane_measurements = self.simulator.step_measurements["lane"]
```

```
        return sum(data["n_queued"] for data in
lane_measurements.values())


class NetworkParser(BaseNetworkParser):
    @property
    @lru_cache(maxsize=None)
    def k_hop_neighbors(self):
        k_hop_neighbors = {}
        max_hops = self.config.misc["max_hops"]
        for ts_id in self.traffic_signal_ids:
            k_hop_neighbors[ts_id] = {}
            for k in range(1, max_hops + 1):
                k_hop_neighbors[ts_id][k] =
self._get_k_hop_neighbors_for_ts(ts_id, k)
        return k_hop_neighbors
```

Лістинг А.7 – Фрагмент реалізації простору дій агентів, фазових обмежень, формування стану світлофорного агента

```
      class PhaseSwitchActionSpace(BaseActionSpace):
    """
    In this child class, actions are binary:
    0 for no switch (remain in the current phase)
    1 for switch (move to the next phase in a round-robin manner).
    """


    def apply(self, actions):
        self._check_actions_type(actions)
        for ts_idx, ts in enumerate(self.traffic_signals.values()):
            current_phase = ts.controller.program.current_phase_index
            if actions[ts_idx] == 1:
                next_phase = (current_phase + 1)% ts.controller.n_phases
                ts.action_to_phase(next_phase)
            else:
                ts.action_to_phase(current_phase)


    def get_size(self):
        return 2
```

ДОДАТОК Б

Конфігураційні параметри SUMO-сценаріїв

Лістинг Б.1 – Конфігурація сценарію 2x2_sumo_grid

```yaml
sumo:
  render: False
  sumo_config_file: 2x2_sumo_grid.sumocfg
  sumo_net_file: 2x2_sumo_grid.net.xml
network:
  network_type: synthetic
signal:
  yellow_time: 5
  min_green_time: 5
  max_green_time: 60
  visibility: 10
  round_robin: True
grid_generator:
  grid_length: 300
  grid_x_number: 2
  grid_y_number: 2
  grid_attach_length: 300
  add_turn_lane: True
```

Лістинг Б.2 – Умови запуску та відтворюваність експериментів

```json
{
  "scenario": "2x2_sumo_grid",
  "duration_seconds": 3600,
  "demand_mode": "generated",
  "traffic_scale_values": [0.25, 0.5, 0.75, 1.0],
  "dqn": {
    "hidden_dim": 256,
    "activation": "leaky_relu",
    "learning_rate": 0.001,
    "epsilon_decay_steps": 10000,
    "total_env_steps": 20000
  },
  "ppo": {
    "hidden_dim": 256,
    "activation": "tanh",
    "learning_rate": 0.0003,
    "rollout_steps": 256,
    "ppo_epochs": 4,
    "target_global_steps": 20000
  },
  "baseline": {
    "controller": "fixed_time",
    "neural_network": false,
    "training": false
  }
}
```

ДОДАТОК В

Структури вхідних даних експериментального середовища

Лістинг В.1 – Фрагмент конфігураційного файлу 2x2_sumo_grid.sumocfg

```
<configuration>
  <input>
    <net-file value="3x3_sumo_grid.net.xml"/>
    <route-files value="3x3_sumo_grid.rou.alt.xml"/>
    <additional-files value="3x3_sumo_grid.add.xml"/>
  </input>
  <time>
    <begin value="0"/>
    <end value="3600"/>
  </time>
  <report>
    <verbose value="true"/>
    <no-step-log value="true"/>
  </report>
</configuration>
```

Лістинг В.2 – Фрагмент файлу додаткових детекторів 2x2_sumo_grid.add.xml

```
<additional xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/addition
al_file.xsd">
  <laneAreaDetector file="e2output.xml" freq="1" friendlyPos="x"
                    id="e2det_A0A1_0" lane="A0A1_0"
                    length="50" pos="222.70000000000002"/>
  <laneAreaDetector file="e2output.xml" freq="1" friendlyPos="x"
                    id="e2det_A0A1_1" lane="A0A1_1"
                    length="50" pos="222.70000000000002"/>
  <laneAreaDetector file="e2output.xml" freq="1" friendlyPos="x"
                    id="e2det_B1A1_0" lane="B1A1_0"
                    length="50" pos="222.70000000000002"/>
</additional>
```

Лістинг В.3 – Фрагмент згенерованого route/sumocfg у директорії результатів generated_routes

```
      <configuration>
  <input>
    <net-file
value="/...//scenarios/sumo/2x2_sumo_grid/2x2_sumo_grid.net.xml"/>
```

```
    <route-files
value="/.../training_runs/.../generated_routes/dqn_2x2_sumo_grid_generated
.rou.xml"/>
  </input>
  <time>
    <begin value="0"/>
    <end value="3600"/>
  </time>
</configuration>
```

ДОДАТОК Г

Оцінювання моделей і збереження метрик

Лістинг Г.1 – Фрагмент запуску оцінювання з файлу run_controllers.py

```
    def run_evaluation(
    scenario,
    simulator_backend,
    controller,
    hours=1,
    add_env_args={},
    add_controller_args={},
    profile=False,
    output_folder=None,
):
    eval_class = Evaluate if controller in non_rl_controllers else
RLEvaluate
    evaluate = eval_class(
        scenario,
        simulator_backend,
        controller,
        add_env_args,
        add_controller_args,
    )
    evaluate.run(hours, save_stats=True, plot_stats=False,
output_folder=output_folder)
    stats = pd.DataFrame(evaluate.log)
    stats["controller"] = controller
    return stats


def save_stats_to_file(all_stats, simulator_backend, scenario,
output_folder):
    fname = os.path.join(output_folder, "stats.csv")
    all_stats.to_csv(fname, index=False)
```

ДОДАТОК Д

Програмна реалізація launcher-а

Лістинг Д.1 – Фрагмент конфігурації DQN з файлу marl_dqn.py

```
@dataclass
class DQNConfig:
    scenario: str = "2x2_sumo_grid"
    simulator_backend: str = "sumo"
    total_env_steps: int = 20000
    warmup_steps: int = 1000
    batch_size: int = 256
    buffer_capacity: int = 50000
    train_frequency: int = 4
    gradient_updates: int = 1
    target_update_interval: int = 500
    hidden_layers: int = 2
    hidden_dim: int = 128
    activation: str = "relu"
    optimizer: str = "adam"
    learning_rate: float = 1e-3
    gamma: float = 0.99
    epsilon_start: float = 1.0
    epsilon_end: float = 0.05
    epsilon_decay_steps: int = 10000
    queue_weight: float = 1.0
    delay_weight: float = 1.0
    switch_penalty: float = 0.1
    eval_interval: int = 2000
    eval_episodes: int = 3
    checkpoint_interval: int = 5000
    demand_mode: str = "original"
    demand_scale: float = 1.0
```

Лістинг Д.2 – Фрагмент буфера досвіду DQN з файлу marl_dqn.py

```
class ReplayBuffer:
    def __init__(self, capacity: int):
        self.buffer: Deque[Tuple[np.ndarray, ...]] = deque(maxlen=capacity
)

    def add(
        self,
        state: np.ndarray,
        action: int,
        reward: float,
        next_state: np.ndarray,
        done: bool,
        action_mask: np.ndarray,
        next_action_mask: np.ndarray,
    ):
        self.buffer.append(
```

```
            (
                state.copy(),
                int(action),
                float(reward),
                next_state.copy(),
                float(done),
                action_mask.copy(),
                next_action_mask.copy(),
            )
        )

    def sample(self, batch_size: int):
        batch = random.sample(self.buffer, batch_size)
        states, actions, rewards, next_states, dones, masks, next_masks =
zip(*batch)
        return {
            "states": torch.tensor(np.asarray(states), dtype=torch.float32
),
            "actions": torch.tensor(np.asarray(actions), dtype=torch.long)
,
            "rewards": torch.tensor(np.asarray(rewards), dtype=torch.float
32),
            "next_states": torch.tensor(np.asarray(next_states), dtype=tor
ch.float32),
            "dones": torch.tensor(np.asarray(dones), dtype=torch.float32),
            "action_masks": torch.tensor(np.asarray(masks), dtype=torch.fl
oat32),
            "next_action_masks": torch.tensor(np.asarray(next_masks), dtyp
e=torch.float32),
        }
```

Лістинг Д.3 – Фрагмент конфігурації та буфера PPO з файлу marl_ppo.py

```
@dataclass
class PPOConfig:
    scenario: str = "2x2_sumo_grid"
    simulator_backend: str = "sumo"
    total_updates: int = 200
    target_global_steps: Optional[int] = None
    rollout_steps: int = 256
    ppo_epochs: int = 4
    minibatch_size: int = 256
    hidden_layers: int = 2
    hidden_dim: int = 128
    activation: str = "relu"
    optimizer: str = "adam"
    learning_rate: float = 3e-4
    gamma: float = 0.99
    gae_lambda: float = 0.95
    clip_epsilon: float = 0.2
    value_coef: float = 0.5
    entropy_coef: float = 0.01
    checkpoint_interval: int = 25
```

```
    reward_mode: str = "local"
    demand_mode: str = "original"
    demand_scale: float = 1.0


class RolloutBuffer:
    def reset(self):
        self.obs: List[np.ndarray] = []
        self.action_masks: List[np.ndarray] = []
        self.actions: List[np.ndarray] = []
        self.log_probs: List[np.ndarray] = []
        self.rewards: List[np.ndarray] = []
        self.values: List[np.ndarray] = []
        self.dones: List[float] = []
        self.returns = None
        self.advantages = None

    def compute_returns_and_advantages(
        self,
        last_values: np.ndarray,
        gamma: float,
        gae_lambda: float,
    ):
        rewards = np.asarray(self.rewards, dtype=np.float32)
        values = np.asarray(self.values, dtype=np.float32)
        dones = np.asarray(self.dones, dtype=np.float32)
        advantages = np.zeros_like(rewards, dtype=np.float32)
        last_advantage = np.zeros_like(last_values, dtype=np.float32)
```

Лістинг Д.4 – Фрагмент генерації попиту з файлу demand.py

```
def generate_sumo_demand(
    scenario,
    output_dir,
    name_prefix="",
    demand_scale=1.0,
    max_vehicles=None,
    depart_start=None,
    depart_end=None,
    rescale_depart_times=False,
    sampling_mode="every_n",
    seed=0,
):
    output_dir = Path(output_dir)
    generated_dir = output_dir / "generated_routes"
    generated_dir.mkdir(parents=True, exist_ok=True)

    scenario_dir, sumocfg_path, net_path = _read_sumo_config(scenario)
    route_path = _resolve_sumocfg_input(sumocfg_path, "route-files")
    if route_path is None:
        raise ValueError(f"No route-files entry in {sumocfg_path}")

    source_tree = ET.parse(route_path)
    source_root = source_tree.getroot()
    source_vehicles = list(source_root.findall("vehicle"))
```

```
    if not source_vehicles:
        raise ValueError(f"No <vehicle> entries found in {route_path}")

    source_depart_min = min(_vehicle_depart(vehicle) for vehicle in source
_vehicles)
    source_depart_max = max(_vehicle_depart(vehicle) for vehicle in source
_vehicles)
    depart_start = source_depart_min if depart_start is None else float(de
part_start)
    depart_end = source_depart_max if depart_end is None else float(depart
_end)
    if depart_end < depart_start:
        raise ValueError("depart_end must be greater than or equal to depa
rt_start")
```

Лістинг Д.5 – Фрагмент обробки tripinfo з файлу tripinfo.py

```
FIELDS = ("duration", "routeLength", "waitingTime", "timeLoss")


def parse_tripinfo_file(path):
    rows = []
    root = ET.parse(path).getroot()
    for tripinfo in root.iter("tripinfo"):
        row = {"file": path.name, "vehicle_id": tripinfo.attrib.get("id",
"")}
        for field in FIELDS:
            row[field] = float(tripinfo.attrib.get(field, 0.0))
        rows.append(row)
    return rows


def summarize_tripinfo_directory(tripinfo_dir, output_dir=None, filename_p
refix=""):
    tripinfo_dir = Path(tripinfo_dir)
    output_dir = Path(output_dir) if output_dir else tripinfo_dir.parent
    filename_prefix = f"{filename_prefix}_" if filename_prefix else ""
    files = sorted(tripinfo_dir.glob("tripinfo_*.xml"))
    rows = []
    for file in files:
        rows.extend(parse_tripinfo_file(file))

    summary = summarize_tripinfo_rows(rows)
    summary["n_files"] = len(files)
    write_csv(output_dir / f"{filename_prefix}tripinfo_summary.csv", [summ
ary], list(summary.keys()))
    write_csv(output_dir / f"{filename_prefix}tripinfo_vehicles.csv", rows
, ["file", "vehicle_id", *FIELDS])
    return summary
```

Лістинг Д.6 – Фрагмент режимів launcher-а з файлу run_once_gui.py

```
ttk.Combobox(
    button_row,
    textvariable=self.start_action_var,
    values=(
        "Run",
        "Train DQN",
        "Train PPO",
        "Train DQN then PPO",
        "Auto tune stages",
    ),
    state="readonly",
    width=20,
).pack(side="left", padx=(0, 8))

if action == "Train DQN":
    self._start_dqn_training()
elif action == "Train PPO":
    self._start_ppo_training()
elif action == "Train DQN then PPO":
    self._start_dqn_then_ppo()
elif action == "Auto tune stages":
    self._start_auto_tuning()
```

Лістинг Д.7 – Фрагмент побудови графіків

```
def _score_tuning_result(self, summary):
    duration = self._metric_float(summary, "duration_mean")
    waiting = self._metric_float(summary, "waitingTime_mean")
    time_loss = self._metric_float(summary, "timeLoss_mean")
    completed = max(1.0, self._metric_float(summary, "n_completed", 1.0))
    return duration + waiting + time_loss + 1000.0 / completed
```

```
  def plot_robustness_comparison_metrics(output_dir):
    output_dir = Path(output_dir)
    plots_dir = output_dir / "plots_ua"
    plots_dir.mkdir(parents=True, exist_ok=True)
    summary, _ = _load_auto_tuning_summary(output_dir)
    if not summary:
        return []

    created = []
    stages = summary.get("stages", [])
    for metric_key, spec in ROBUSTNESS_METRICS.items():
        series = _robustness_series(stages, metric_key)
        if len(series) < 2:
            continue
```

```
        path = plots_dir /
f"robustness_compare_{spec['order']}_{spec['slug']}.png"
        _save_line_plot(path, spec["title"], "Traffic demand Scale",
spec["y_label"], series)
        created.append(path)
    return created
```

ДОДАТОК Е

Результати підсумкових матеріалів auto-tuning

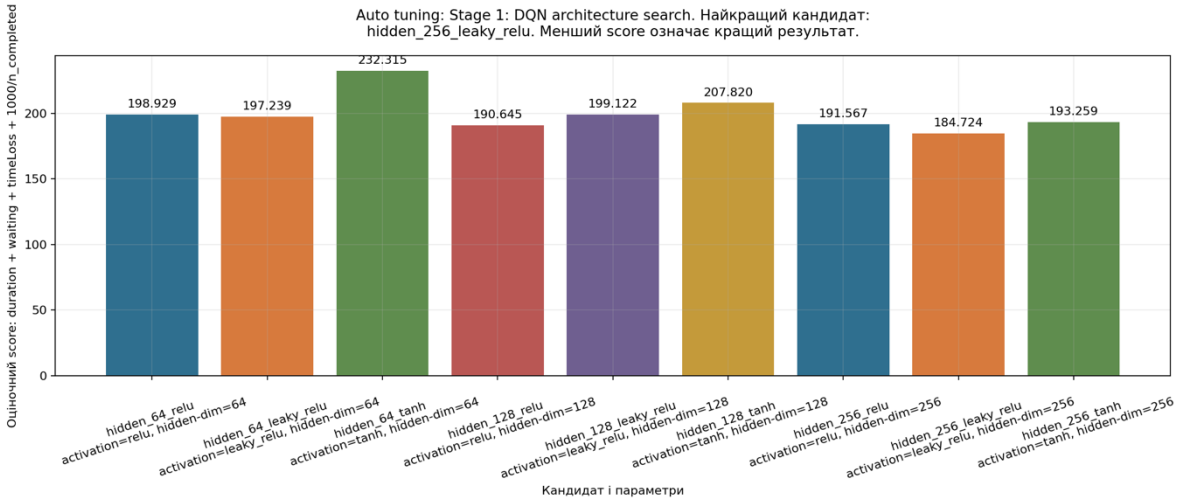


Рисунок Е.1 – Порівняння архітектур DQN-моделі під час автоматизованого підбору параметрів

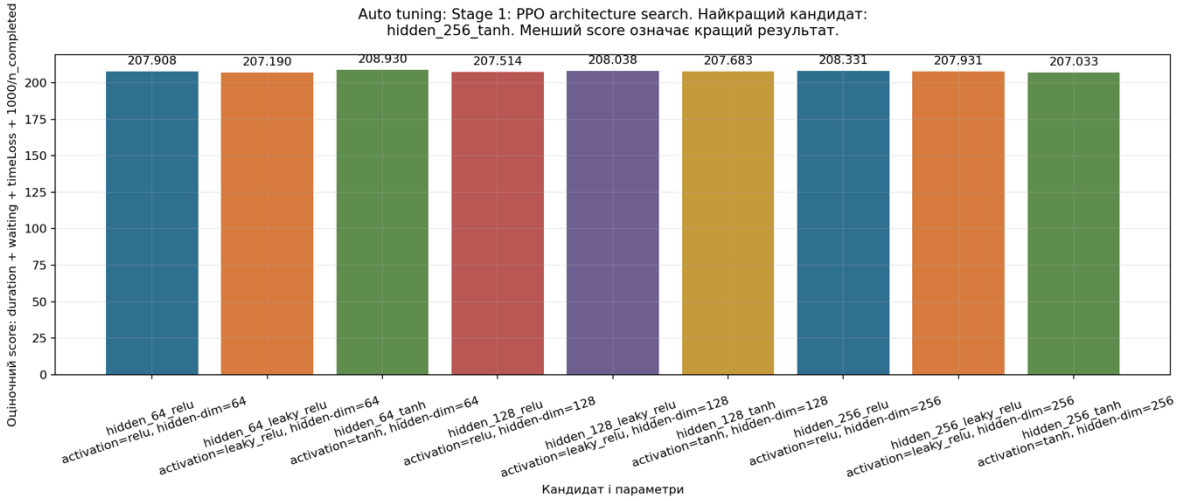


Рисунок Е.2 – Порівняння архітектур PPO-моделі під час автоматизованого підбору параметрів

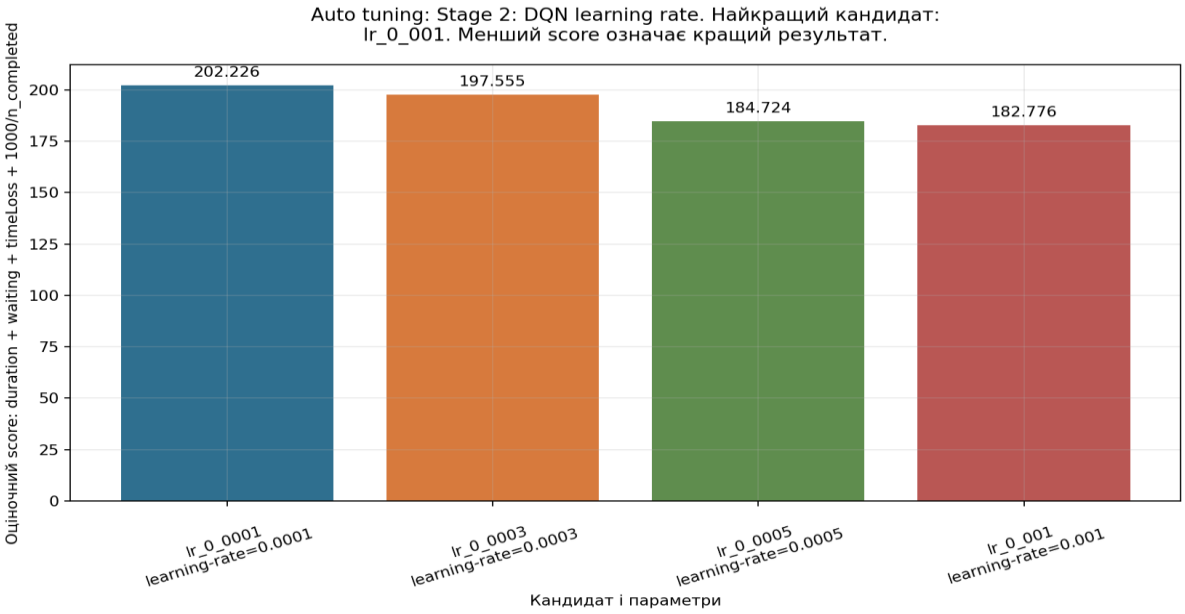


Рисунок Е.3 – Вплив коефіцієнта швидкості навчання на якість роботи DQN-агента

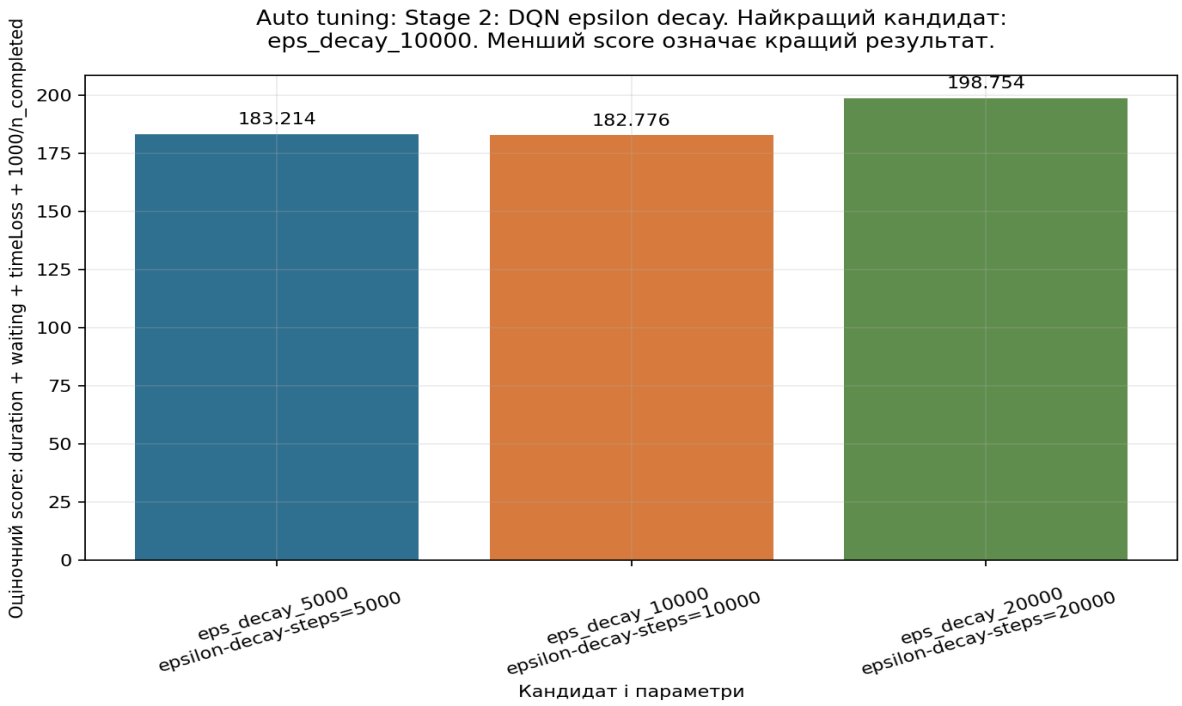


Рисунок Е.4 – Вплив параметра зменшення ε-жадібної стратегії на результати навчання DQN-агента

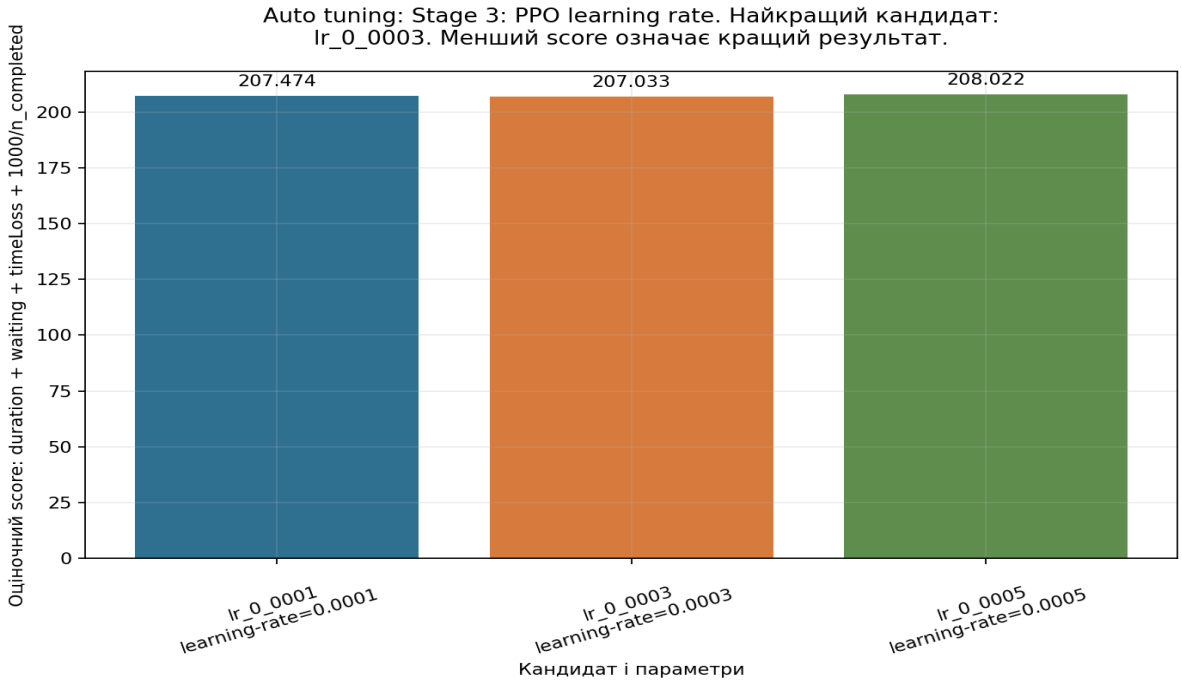


Рисунок Е.5 – Вплив коефіцієнта швидкості навчання на якість роботи PPO-агента

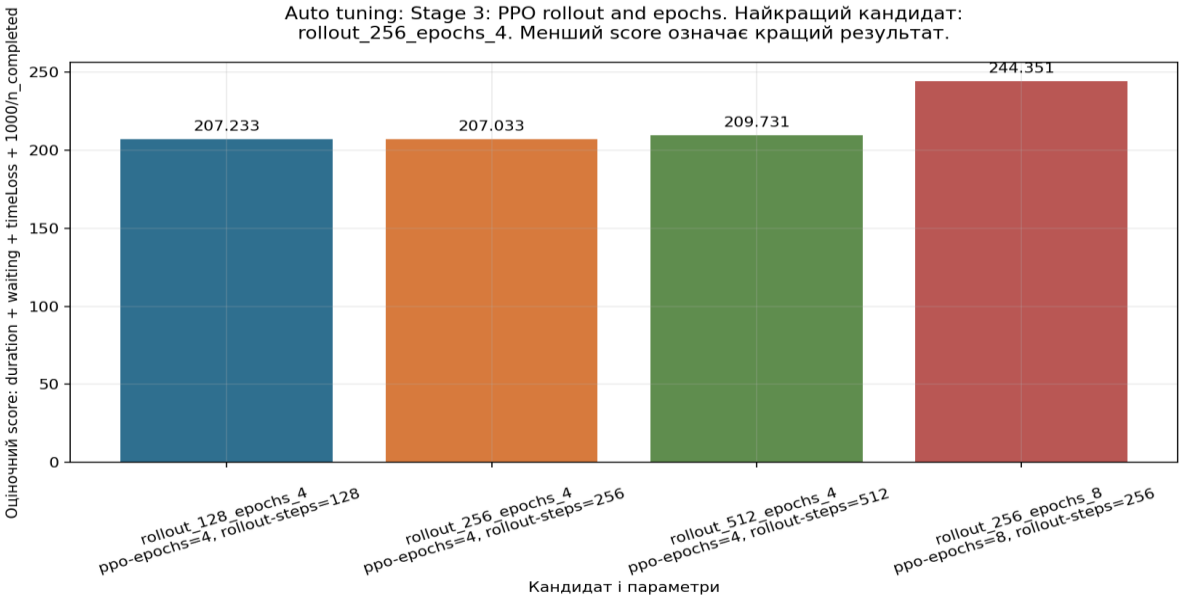


Рисунок Е.6 – Вплив кількості епох оновлення rollout-вибірки на результати навчання PPO-агента

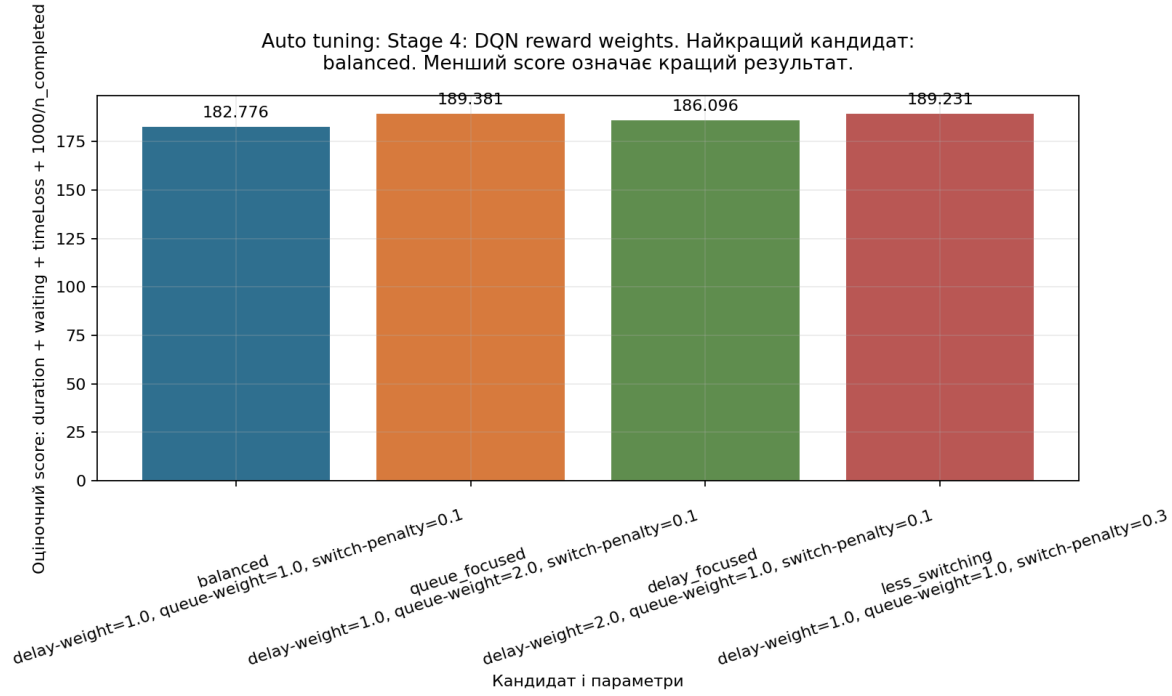


Рисунок Е.7 – Порівняння варіантів вагових коефіцієнтів функції винагороди DQN-агента

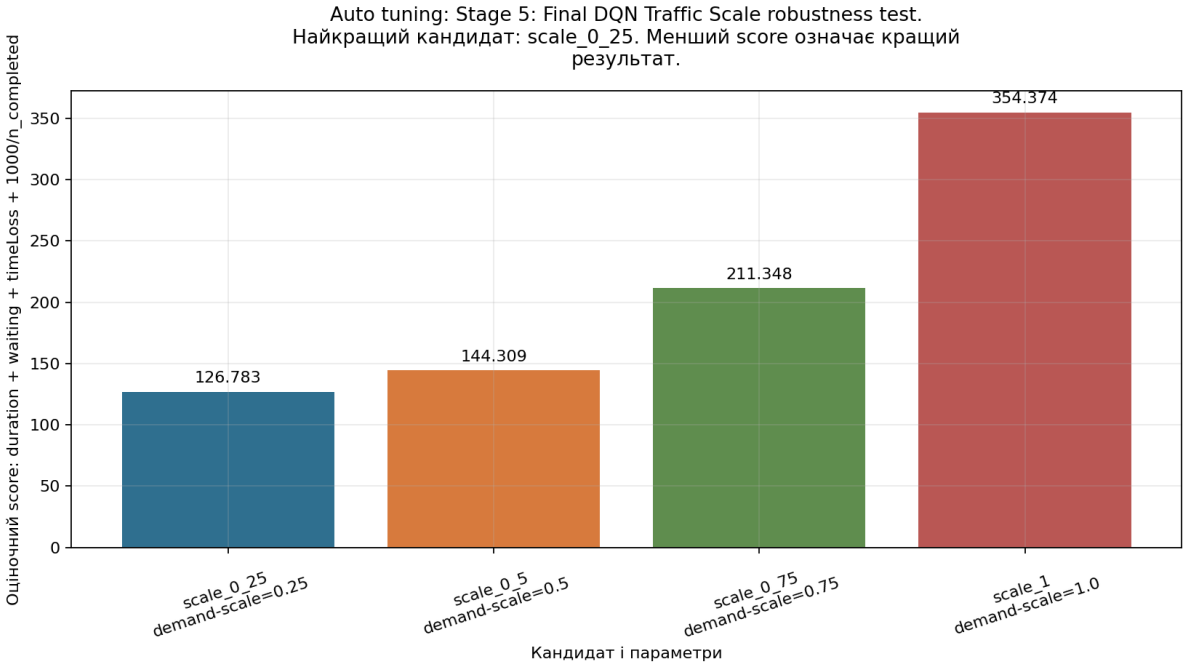


Рисунок Е.8 – Перевірка стійкості DQN-агента за різних рівнів транспортного попиту

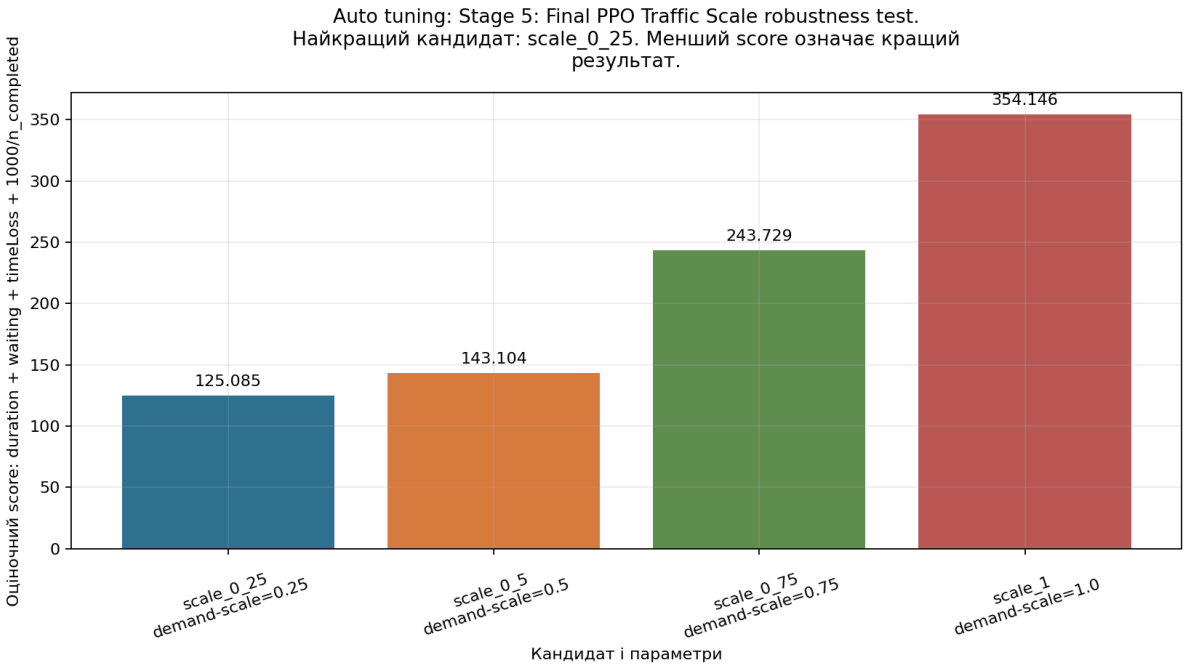


Рисунок Е.9 – Перевірка стійкості PPO-агента за різних рівнів транспортного попиту

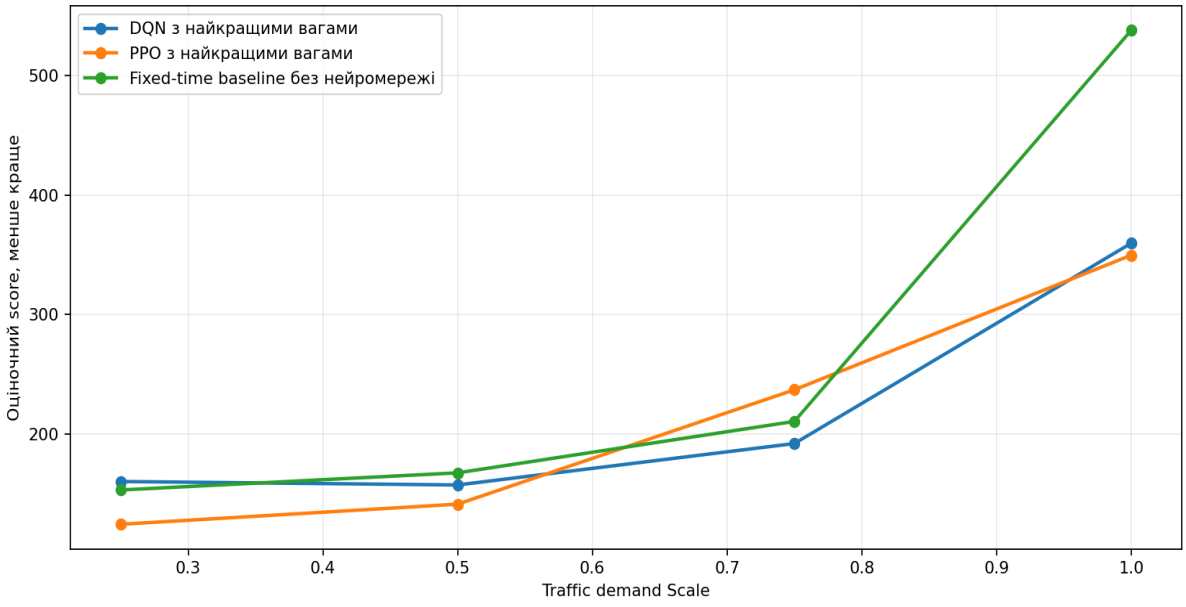


Рисунок Е.10 – Порівняння інтегрального показника score для DQN, PPO та fixed_time за різних значень Traffic Scale

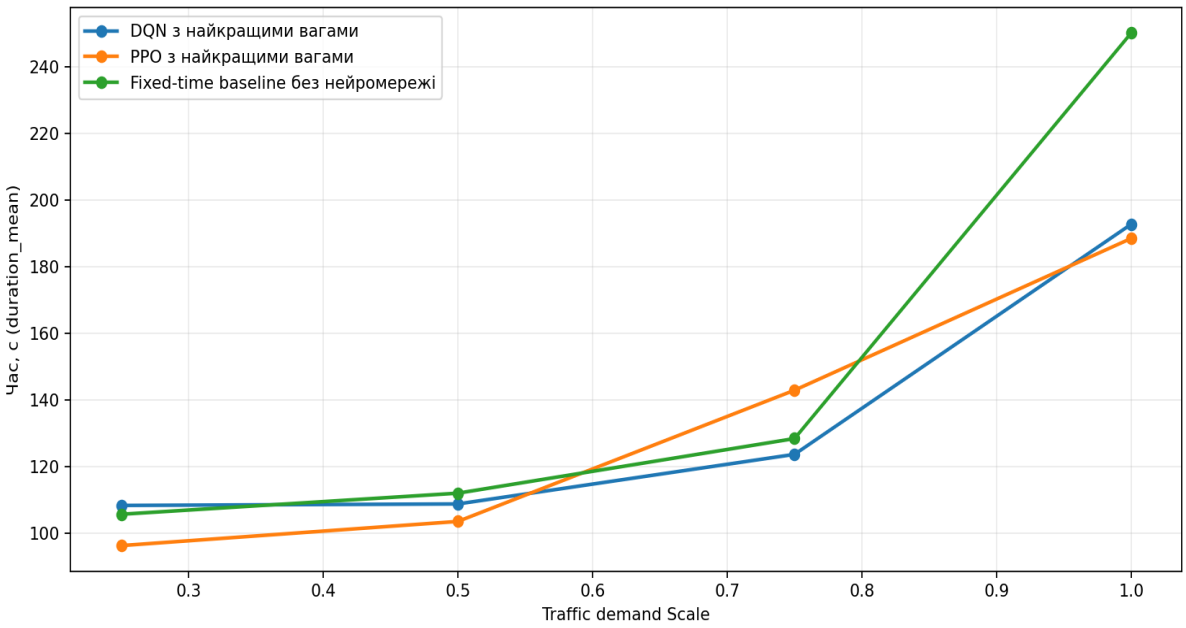


Рисунок Е.11 – Порівняння середнього фактичного часу поїздки для DQN, PPO та fixed_time за різних значень Traffic Scale

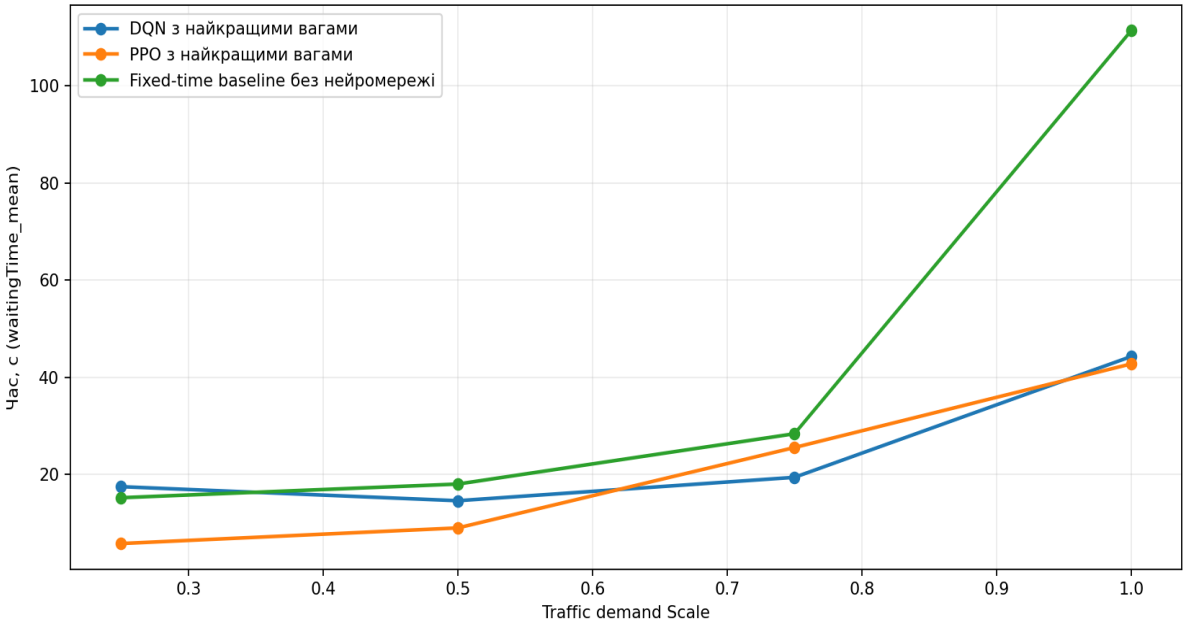


Рисунок Е.12 – Порівняння середнього часу очікування для DQN, PPO та fixed_time за різних значень Traffic Scale

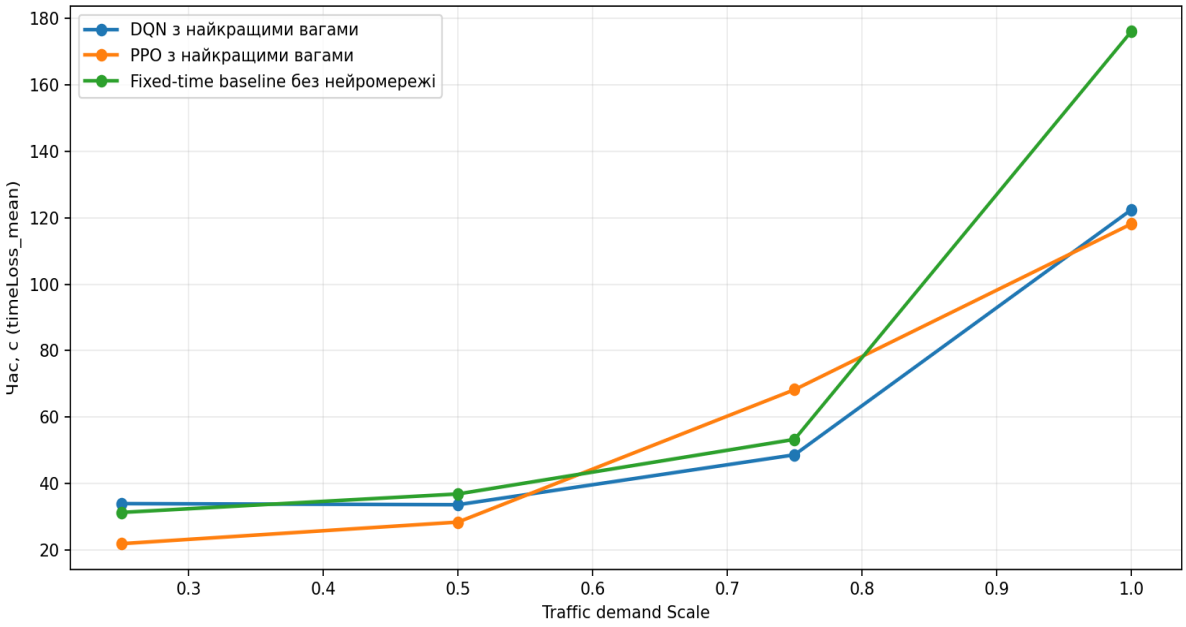


Рисунок Е.13 – Порівняння середніх втрат часу для DQN, PPO та fixed_time за різних значень Traffic Scale

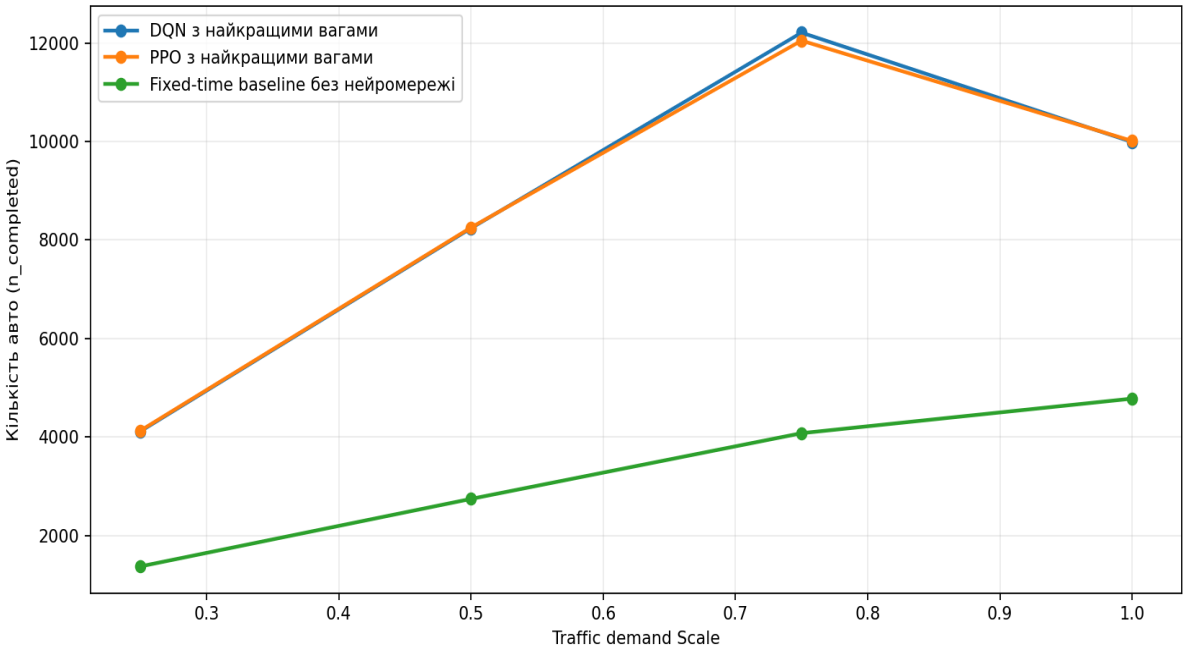


Рисунок Е.14 – Порівняння кількості завершених поїздок для DQN, PPO та fixed_time за різних значень Traffic Scale

ДОДАТОК Ж

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	245 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Колій О.С.			Дослідження методів розробки мультиагентної системи навчання світлофорного регулювання для оптимізації міських транспортних потоків	Шифр групи	Код напр./спец.
Перевірив		Кудрявцева М.С.				СШІздм-24-1	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Чала Л.Е.					