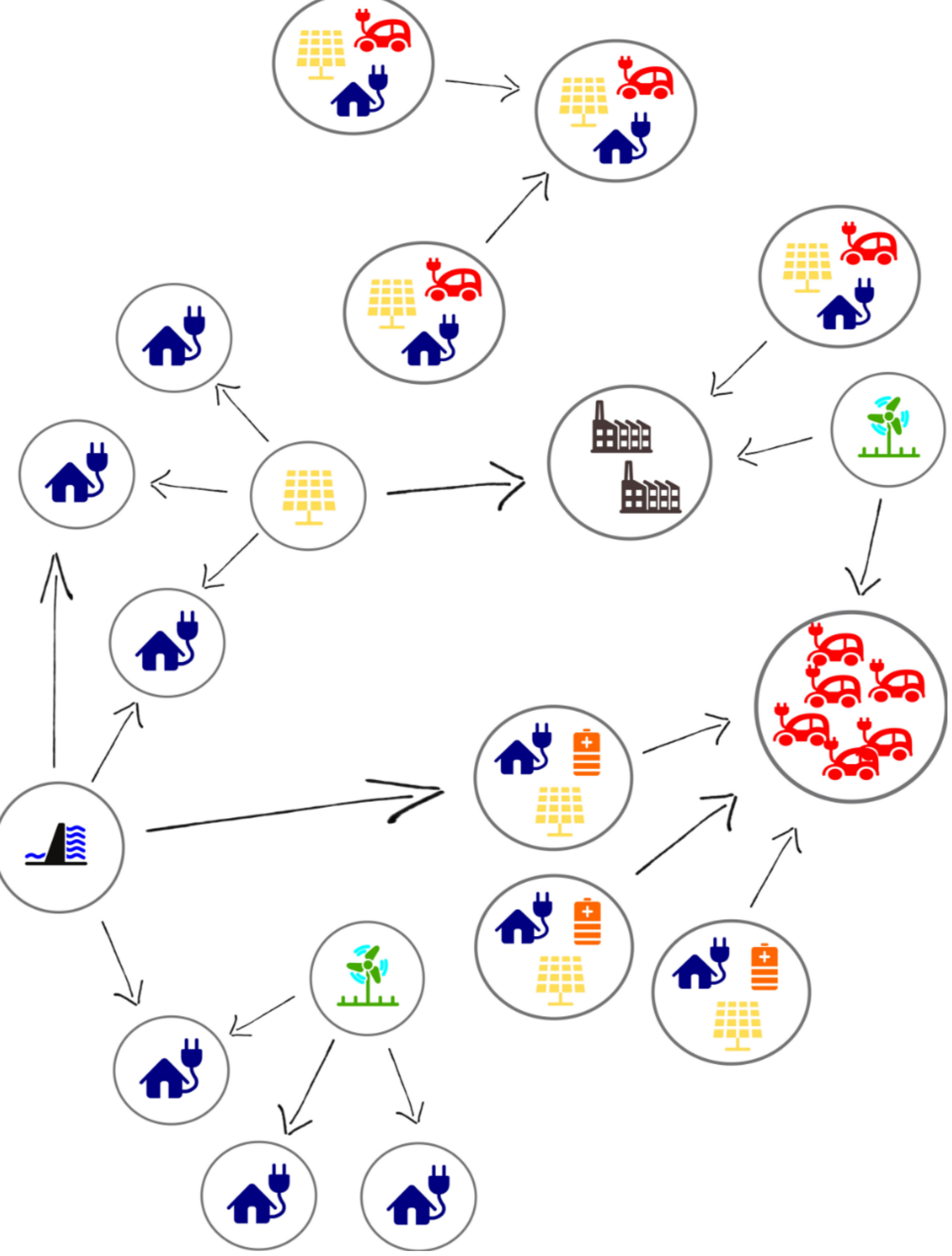


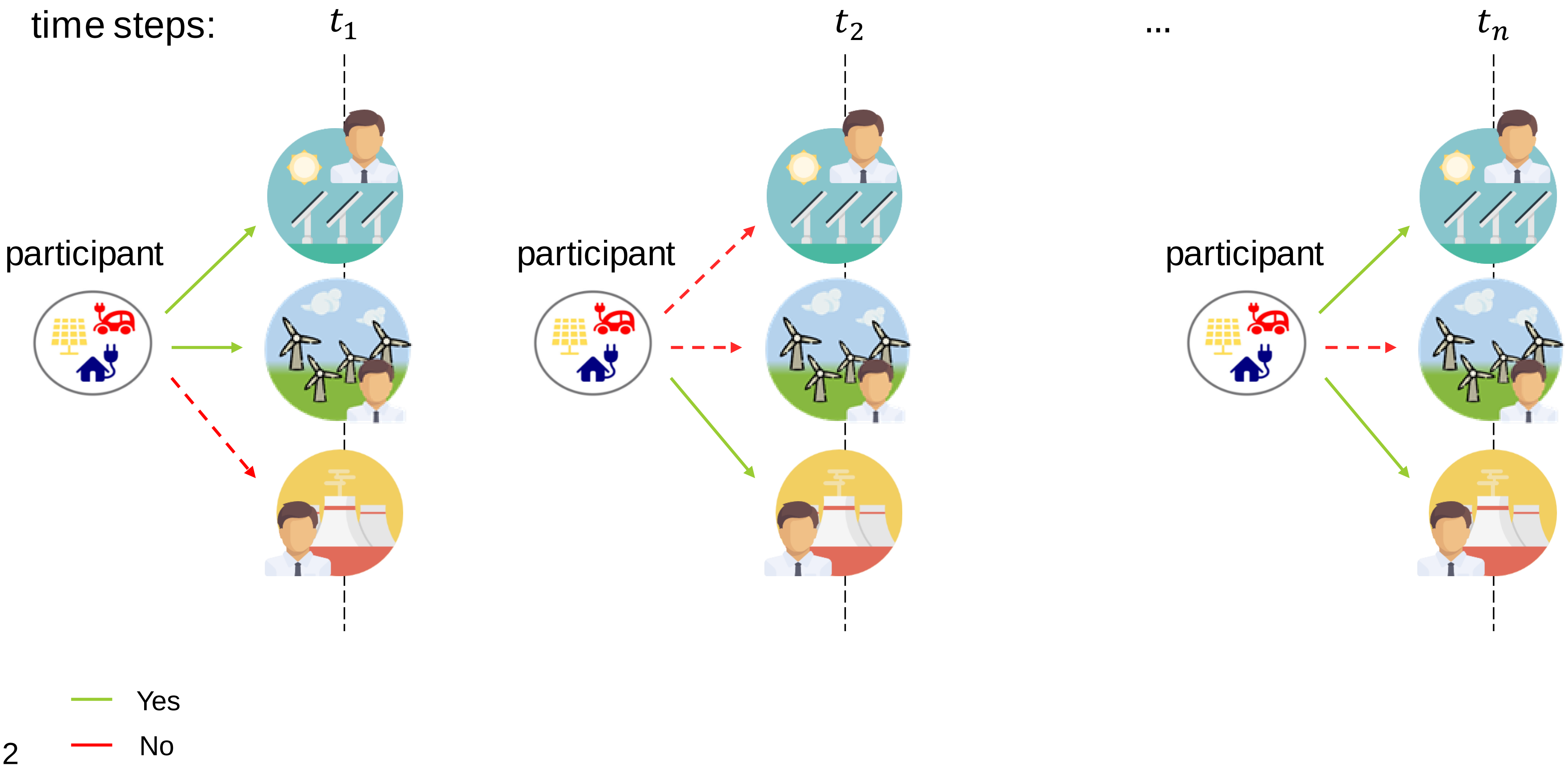
Danske Bank

Reinforcement Learning applications for Energy Analytics and Markets

Tiago Sousa, Pierre Pinson

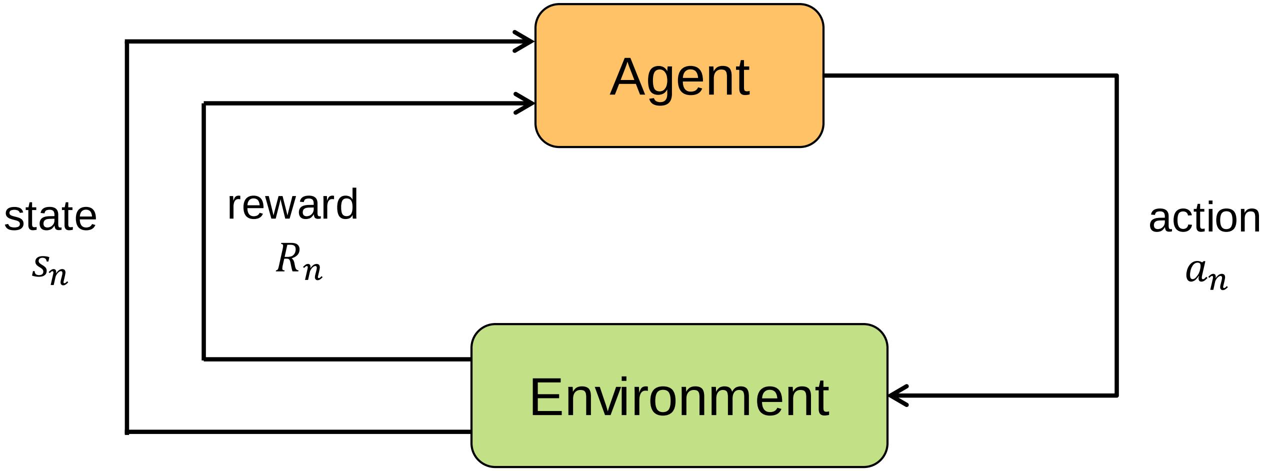
Work done as Postdoc researcher at DTU

Future Energy markets



Motivation

- We can design Energy markets as Reinforcement Learning

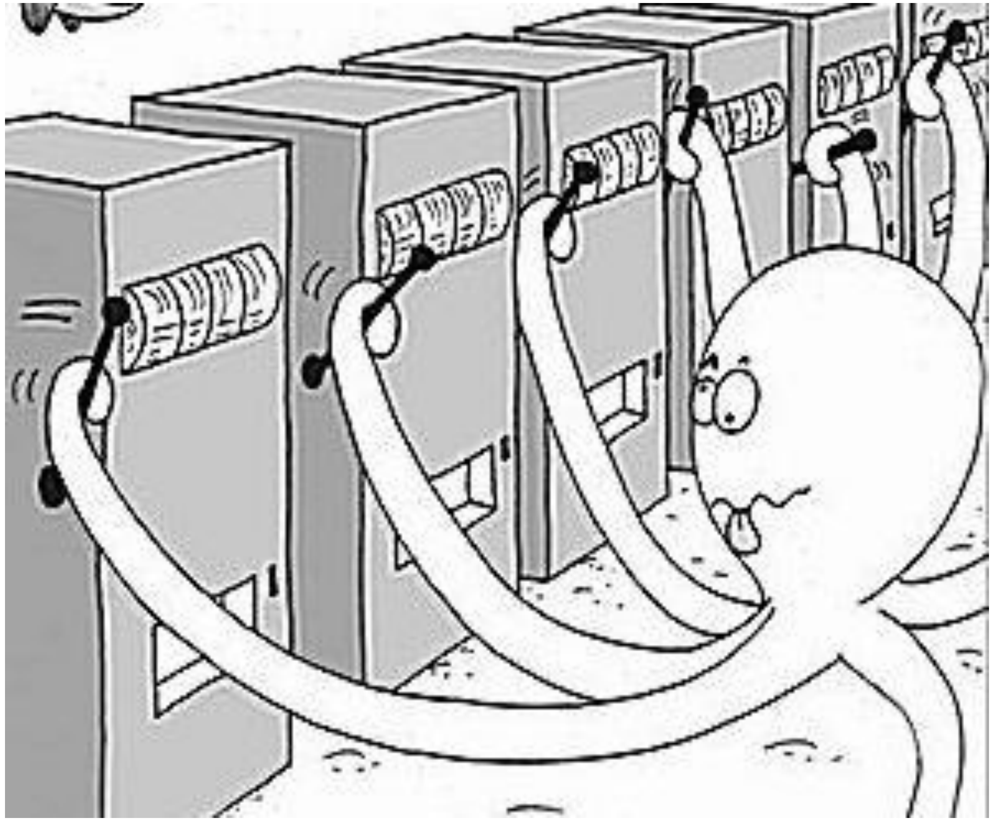


Outline

- Reinforcement Learning approach
  - Multi-Armed Bandit
- Test case and results
- Conclusions and next steps

Multi-Armed Bandit

- Agent learns which arm returns the highest payoff



Real-world applications:

- Clinical trials
- Online Advertising
- Network routing

Danske Bank

Multi-Armed Bandit

Agent

participant

Step n	***Arm 1***	***Arm 2***	***Arm 3***
1	1	0	0
2	0	1	0
3	1	0	0
4	0	0	1

actions a_n

Step *n* ≠ time *t*

Multi-Armed Bandit

Agent

participant

Environment

Step n	Arm 1	Arm 2	Arm 3	Reward
1	1	0	0	1
2	0	1	0	0
3	1	0	0	0
4	0	0	1	1

Total reward

$$R_{Total} = \mathbb{E}(R_n) = \frac{1}{N}\sum R_n$$

$$a_n^* = \text{argmax}\, R_{Total}$$

Multi-Armed Bandit

Agent

participant

Environment

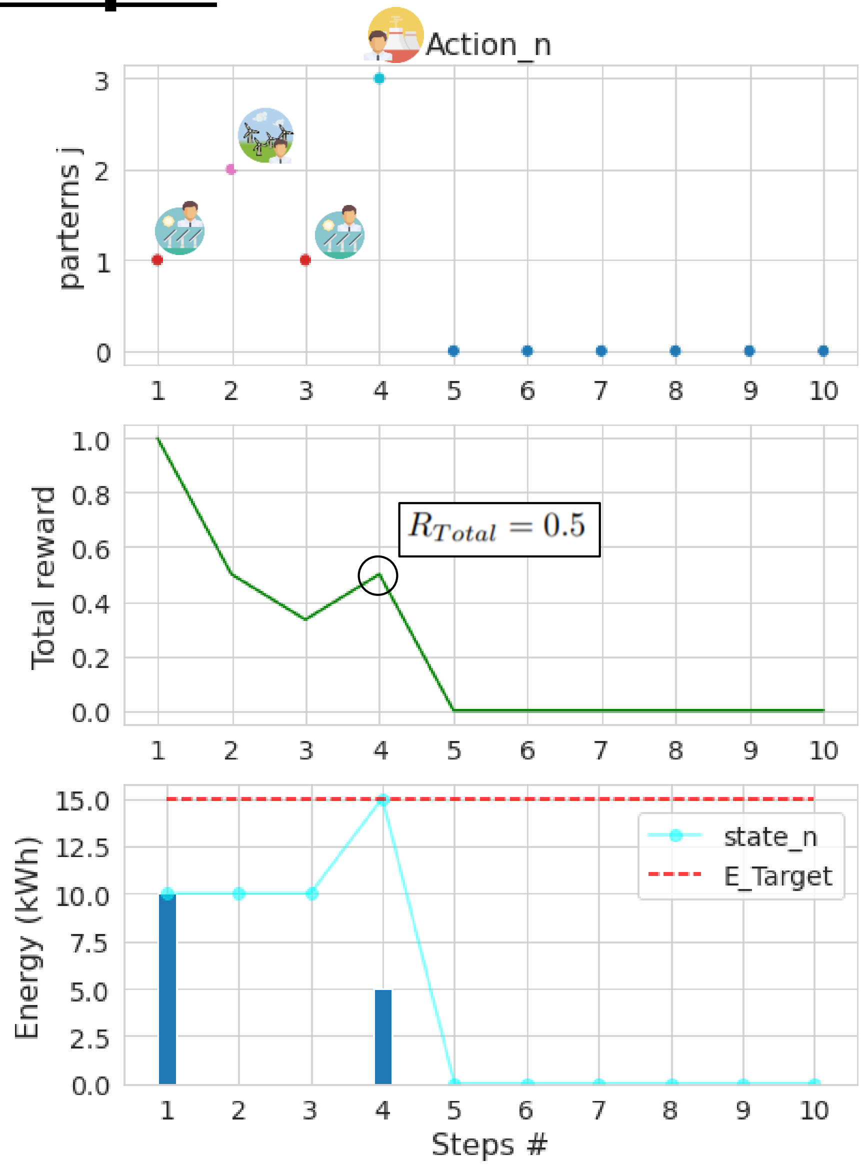
Step n	***Arm 1***	***Arm 2***	***Arm 3***	***Reward***	***Energy (kWh)***
1	1	0	0	1	10
2	0	1	0	0	0
3	1	0	0	0	0
4	0	0	1	1	5

State per step *n*

$$s_n = \sum E_n(j) R_n$$

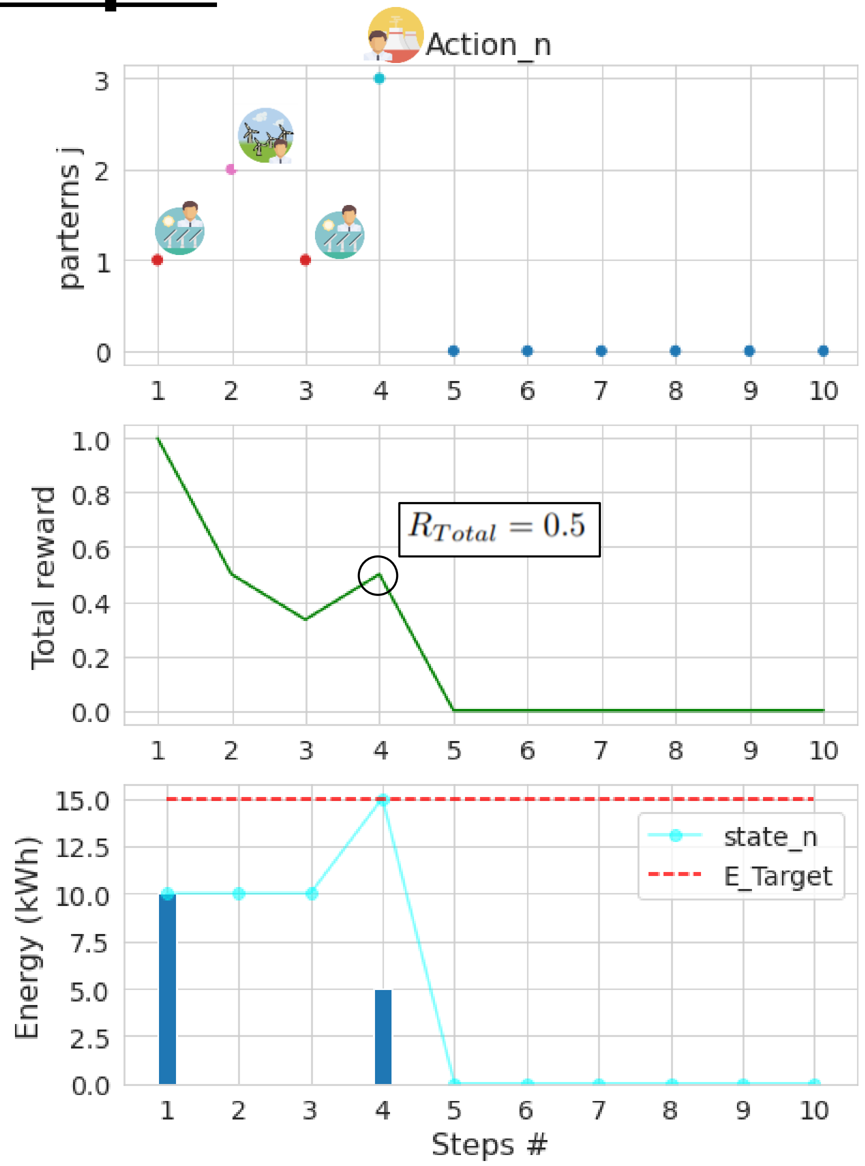
$$s_n \leq E_{Target}$$ Stopping condition

Example



Action_n
parterns j
Total reward
$R_{Total} = 0.5$
Energy (kWh)
state_n
E_Target
Steps #

Example

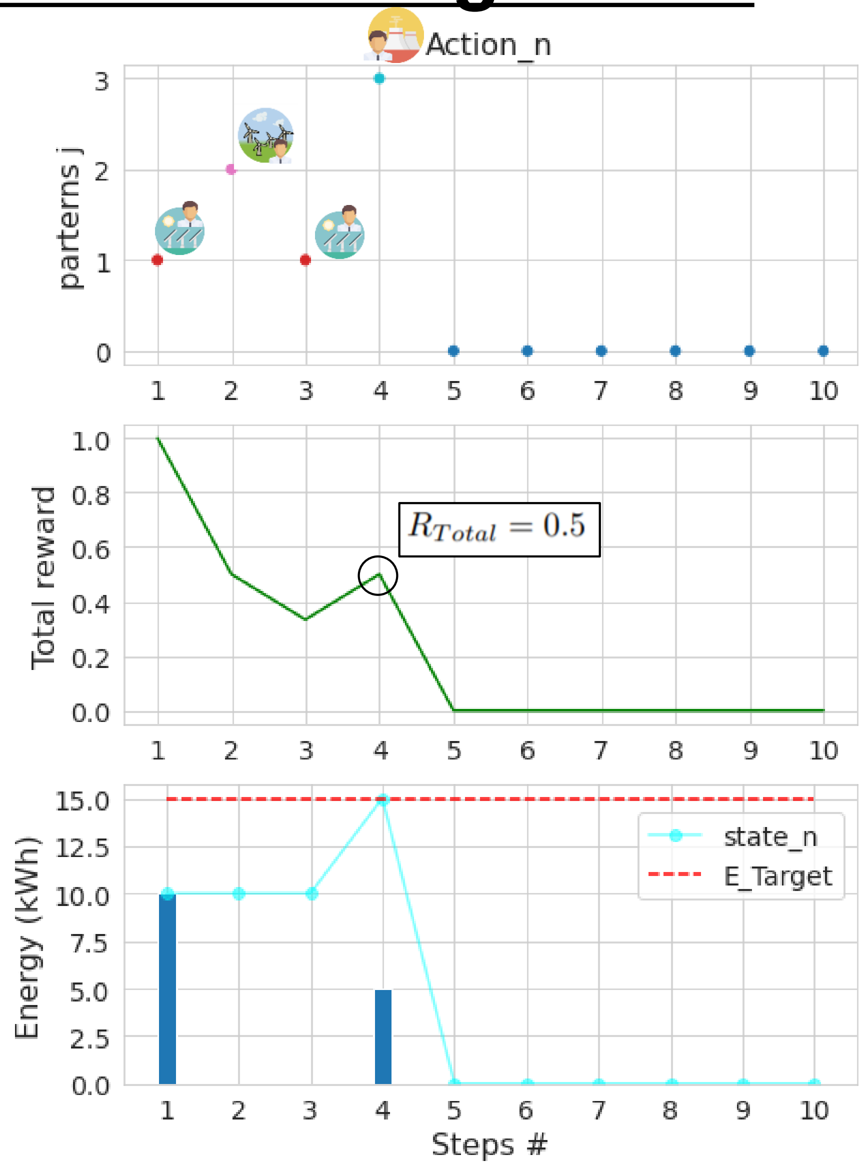


- This iterative process is an episode
- We terminate when
  - $s_n = E_{Target}$

episode = time *t*

Danske Bank

Translate as Algorithm

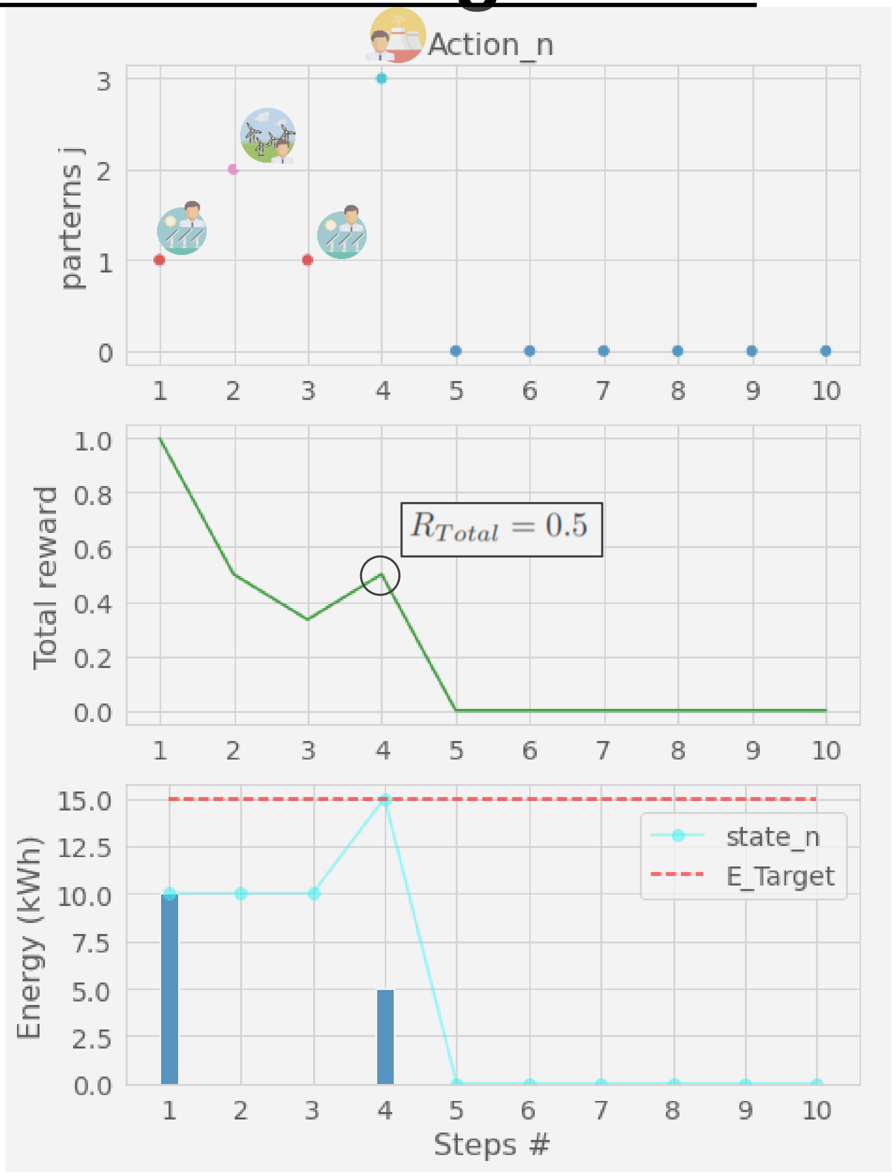


Algorithm for each episode:

Algorithm 1: RL Cycle for each episode

$E_{Target} \leftarrow$ random sample from $[\underline{E}_{Target}, \overline{E}_{Target}]$;
Initialize step $n \leftarrow 1$;
while $s_n \leq E_{Target}$ **do**
    Take action $a_n \leftarrow Arm\ j$ (using policy strategy);
    Observe $R_{Total} \leftarrow \mathbb{E}(R_n)$;
    Update $s_n \leftarrow \sum E_n(a_n) R_n$;
    $n \leftarrow n + 1$;
end

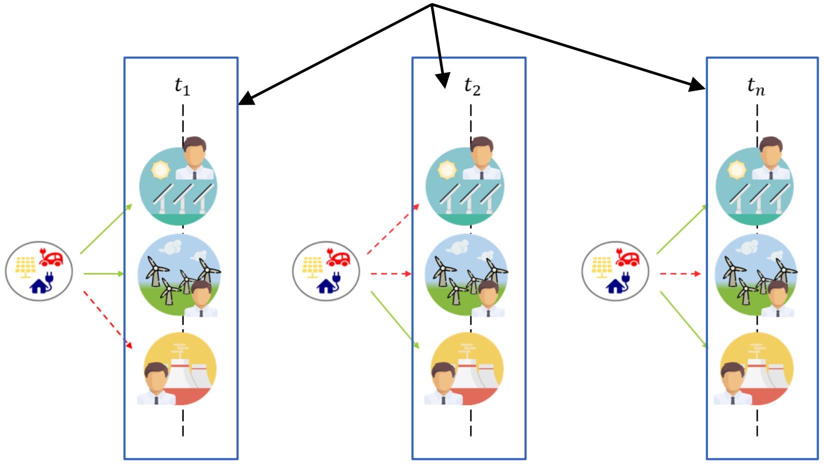
Translate as Algorithm



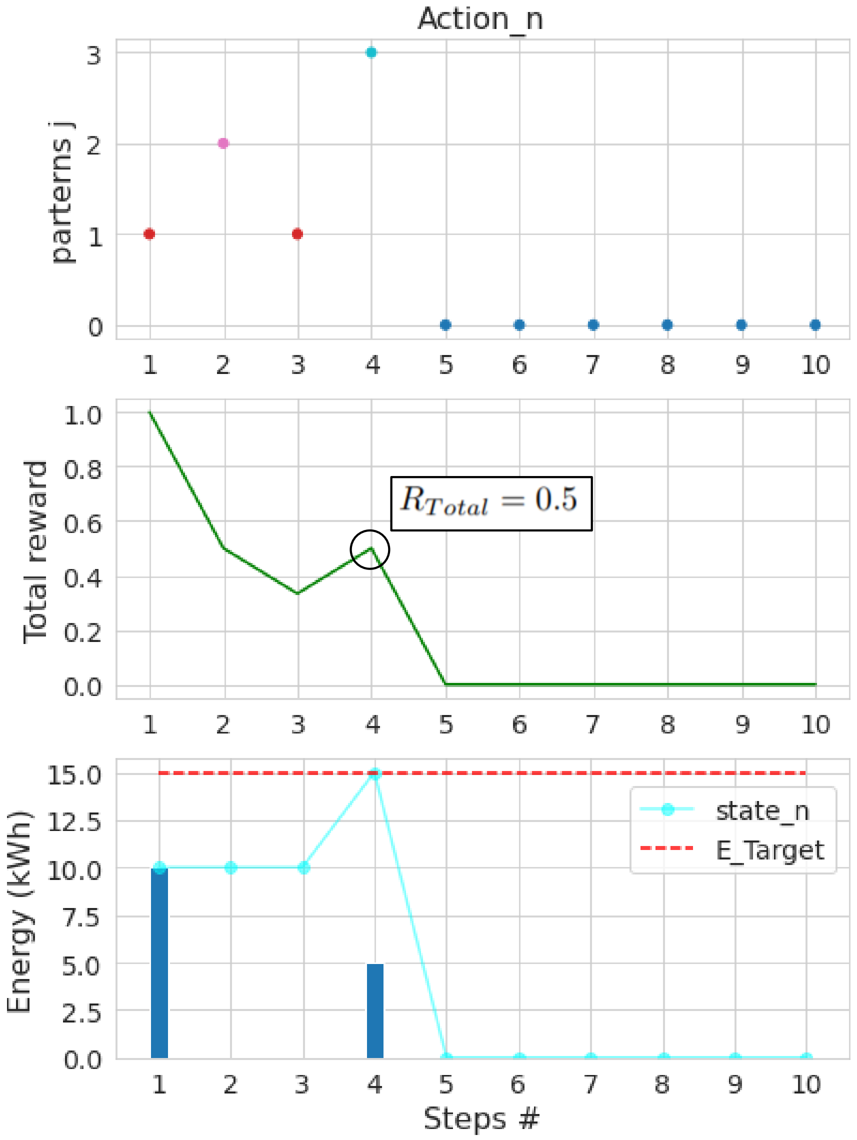
Algorithm for each episode:

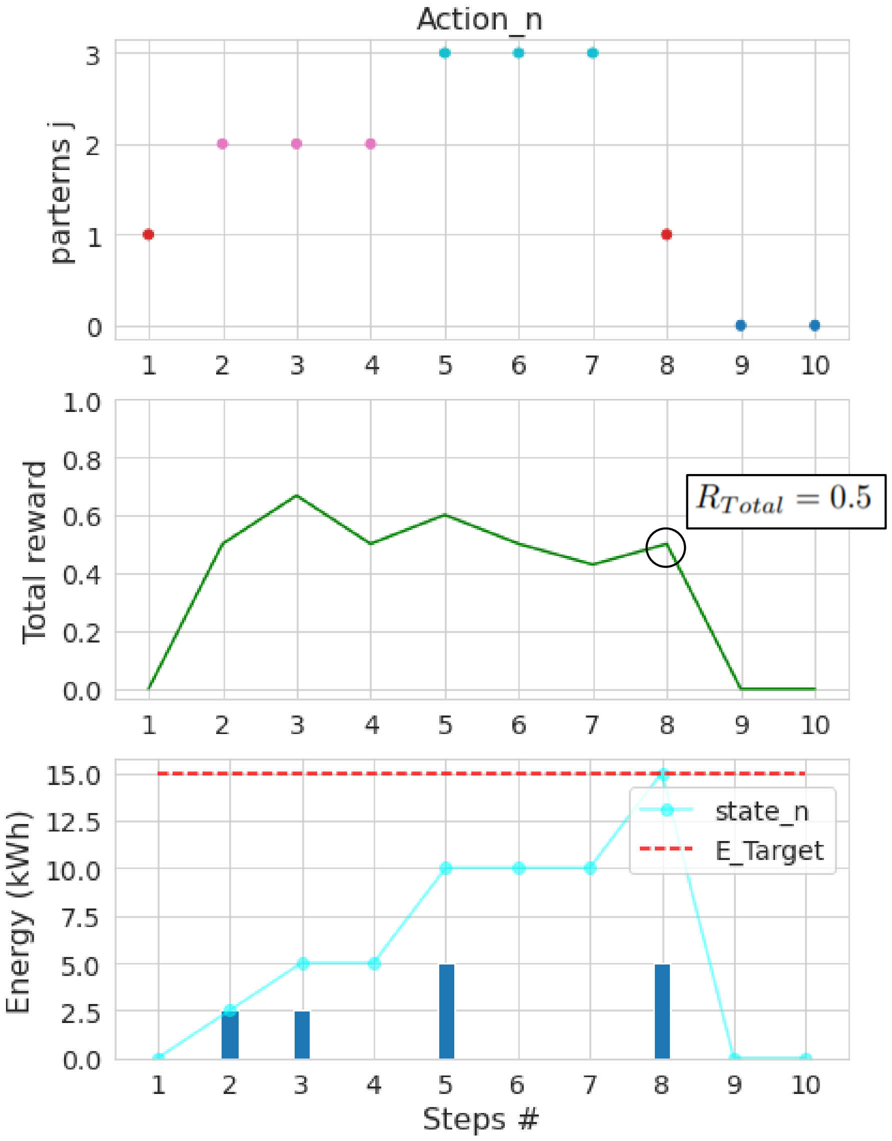
Algorithm 1: RL Cycle for each episode

$E_{Target} \leftarrow$ random sample from $[\underline{E}_{Target}, \overline{E}_{Target}]$;
Initialize step $n \leftarrow 1$;
while $s_n \leq E_{Target}$ **do**
  Take action $a_n \leftarrow Arm\ j$ (using policy strategy);
  Observe $R_{Total} \leftarrow \mathbb{E}(R_n)$;
  Update $s_n \leftarrow \sum E_n(a_n) R_n$;
  $n \leftarrow n + 1$;
end



How to differentiate between episodes?





Multi-Armed Bandit

- Reward is a random variable

Bernoulli distribution

$$R_n(j) \sim \mathrm{B}(1, p_j)$$

For a large number of steps *n*:

$$R_n(j) \approx p_j$$

Environment

Step n	**Arm 1**	**Reward**
1	1	1
2	0	-
3	1	0
4	0	-

Multi-Armed Bandit

- Reward is a random variable

Bernoulli distribution

$$R_n(j) \sim \mathrm{B}(1, p_j)$$

For a large number of steps *n*:

$$R_n(j) \approx p_j$$

Step n	*Arm 1*
1	1
2	0
3	1
4	0

Environment

Reward
1
-
0
-

Action-Value function

Estimator of p_j for every arm j

$$Q_n(j) = \frac{1}{N_j} \sum R_n(j) = \hat{p}_n(j)$$

Danske Bank

Mathematical Formulation

Algorithm 2: Complete algorithm

Initialize episodes $e \in \mathbb{E}$, steps $n \in \mathbb{N}$, actions $a_n \in arms\ \mathbb{J}$;
for *each episode* e **do**
- $E_{Target} \leftarrow$ random sample from $[\underline{E}_{Target}, \overline{E}_{Target}]$;
- Initialize step $n \leftarrow 1$;
- **while** $s_n \leq E_{Target}$ **do**
  - Take action $a_n \leftarrow Arm\ j$ (using policy strategy);
  - Observe $R_{Total} \leftarrow \mathbb{E}(R_n)$;
  - Update $s_n \leftarrow \sum E_n(a_n)R_n$;
  - $n \leftarrow n + 1$;
  - Update every $Q_n(j) \leftarrow \mathbb{E}(R_n(j)) = \hat{p}_n(j)$;
- **end**
- Propagate to the next episode $Q^{e+1}(j) \leftarrow \mathbb{E}(Q^e(j))$;

end

Danske Bank

Mathematical Formulation

Algorithm 2: Complete algorithm

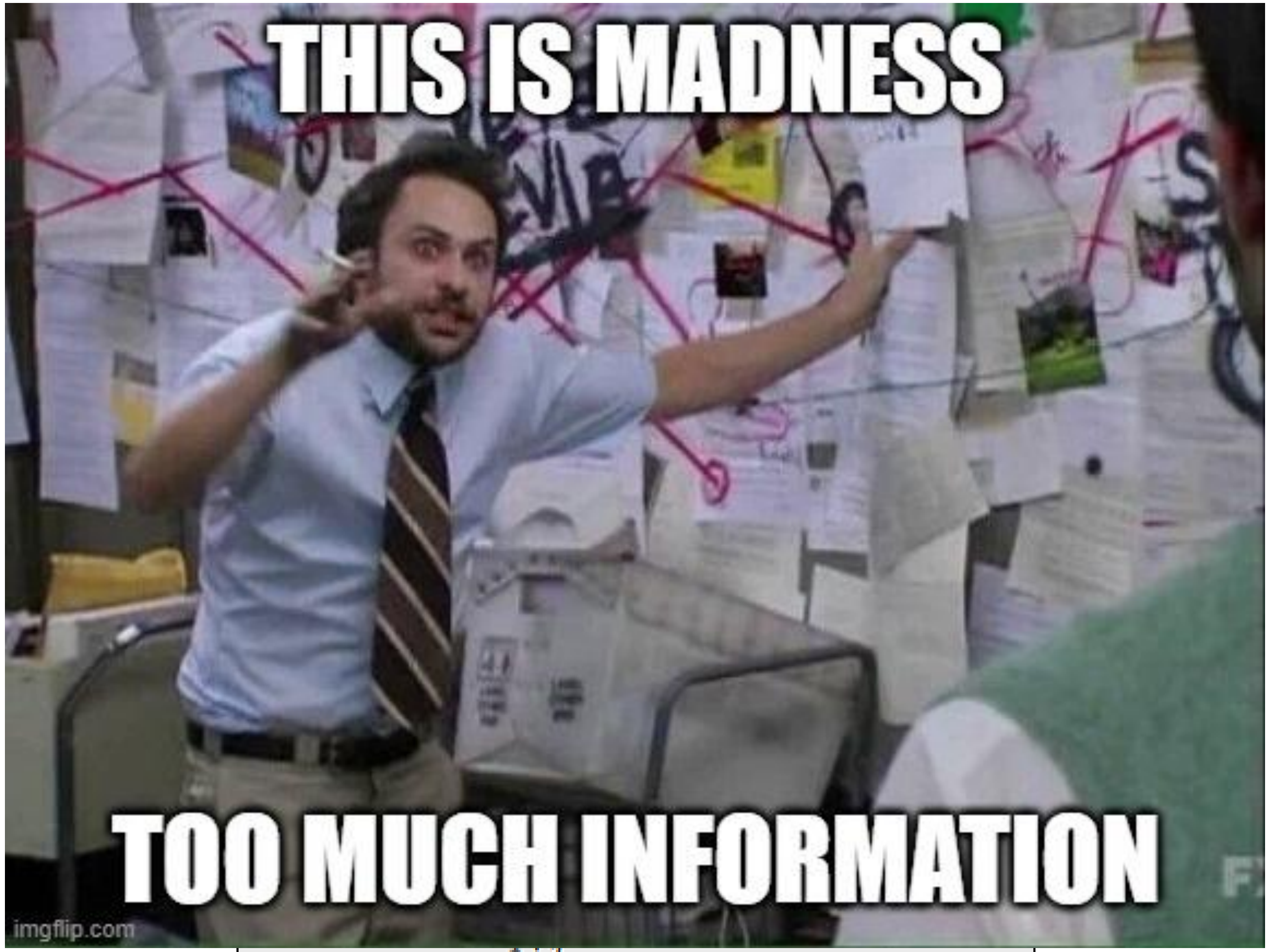
Initialize episodes $e \in \mathbb{E}$, steps $n \in \mathbb{N}$, actions $a_n \in arms\ \mathbb{J}$;
for *each episode* e **do**
  $E_{Target} \leftarrow$ random sample from $[\underline{E}_{Target}, \overline{E}_{Target}]$;
  Initialize step $n \leftarrow 1$;
  while $s_n \leq E_{Target}$ **do**
    Take action $a_n \leftarrow Arm\ j$ (using policy strategy);
    Observe $R_{Total} \leftarrow \mathbb{E}(R_n)$;
    Update $s_n \leftarrow \sum E_n(a_n) R_n$;
    $n \leftarrow n + 1$;
    Update every $Q_n(j) \leftarrow \mathbb{E}(R_n(j)) = \hat{p}_n(j)$;
  end
  Propagate to the next episode $Q^{e+1}(j) \leftarrow \mathbb{E}(Q^e(j))$;
end

Experience replay as NN

- We start with *batch* of episodes with **no propagation**
- After, we compute the average Action-value for each arm *j*

More to say!!!

- How to adopt other propagation strategies between episodes
- Learning algorithms estimate the Action-Value function $Q_n(j)$
  - Random
  - ϵ-greedy
  - Thompson Sampler
  - Upper Confidence Bound

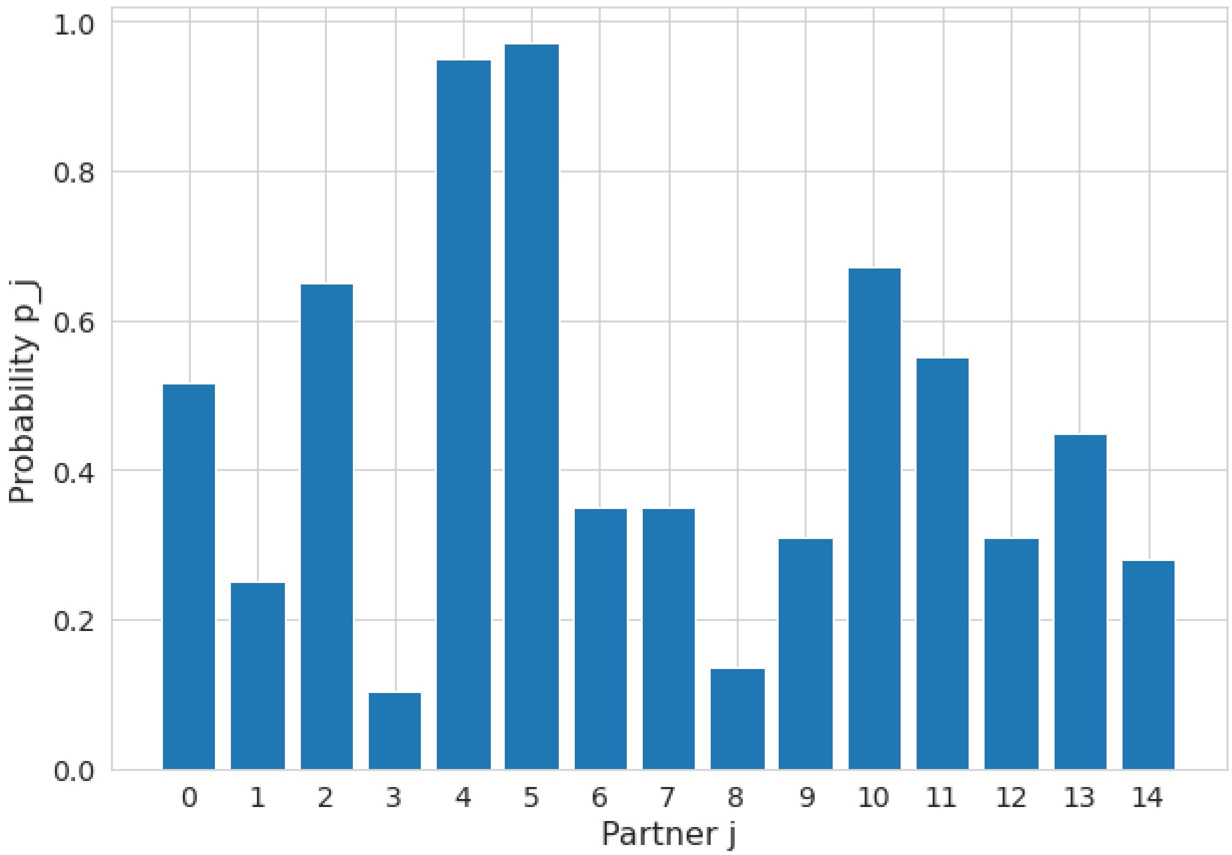


Test case

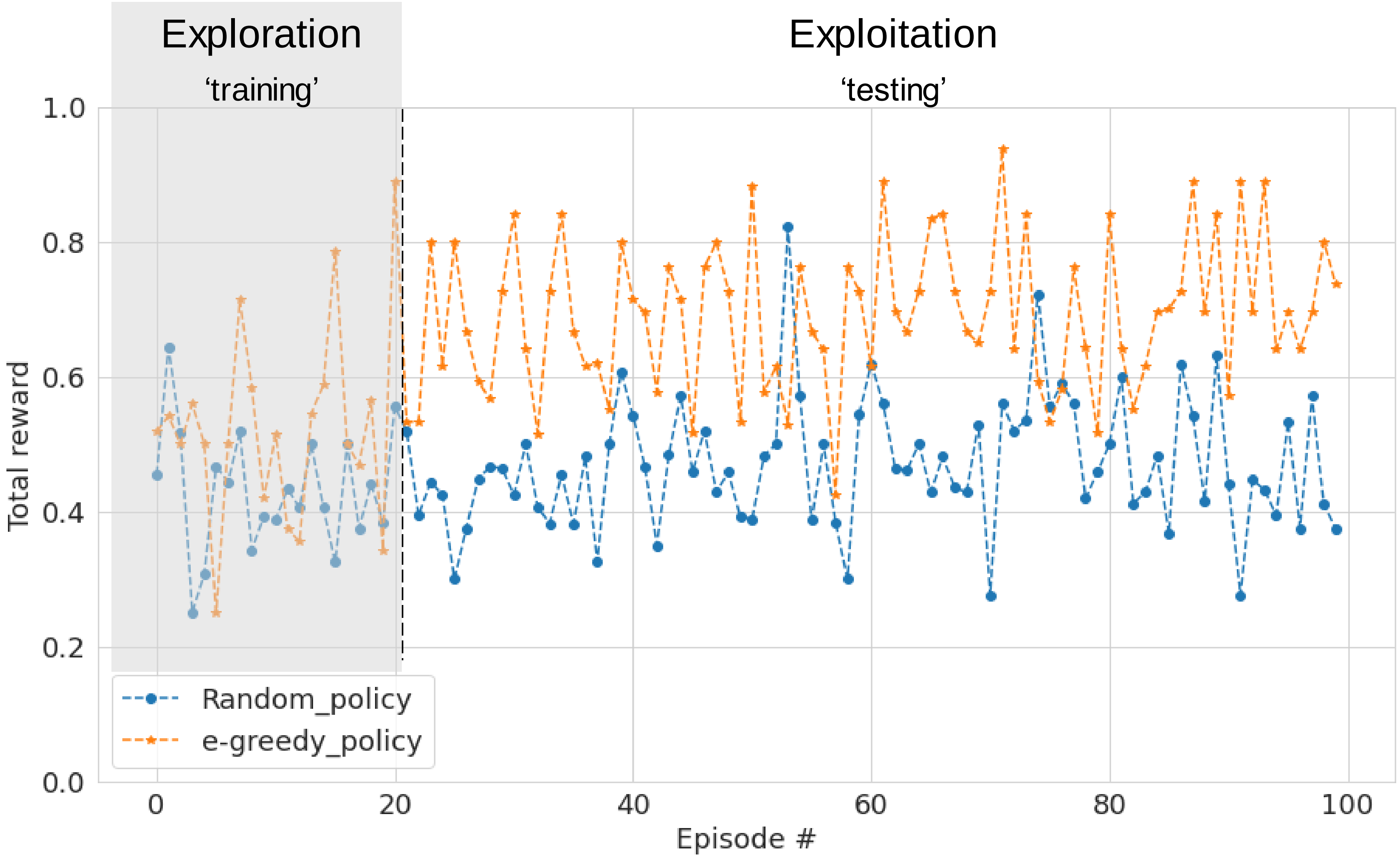
- Case with 15 parterns j

- We used 100 episodes
  - $E_{Target} = 15 \text{ kWh}$

- Learning algorithms:
  - Random
  - ϵ-greedy
  - Thompson Sampler



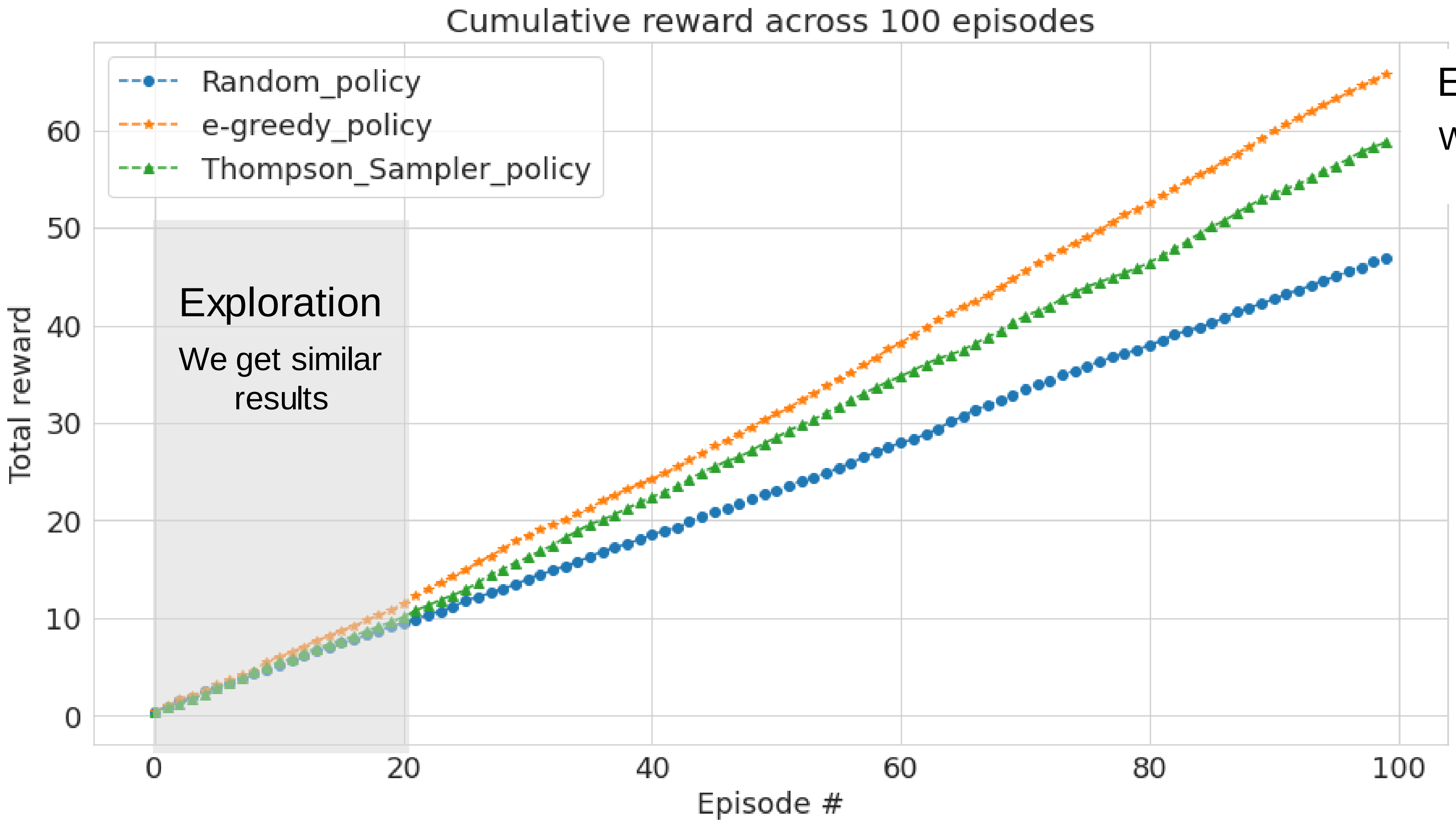
Test case



Exploration
'training'
Exploitation
'testing'
1.0
0.8
0.6
0.4
0.2
0.0
Total reward
0
20
40
60
80
100
Episode #
Random_policy
e-greedy_policy

Danske Bank

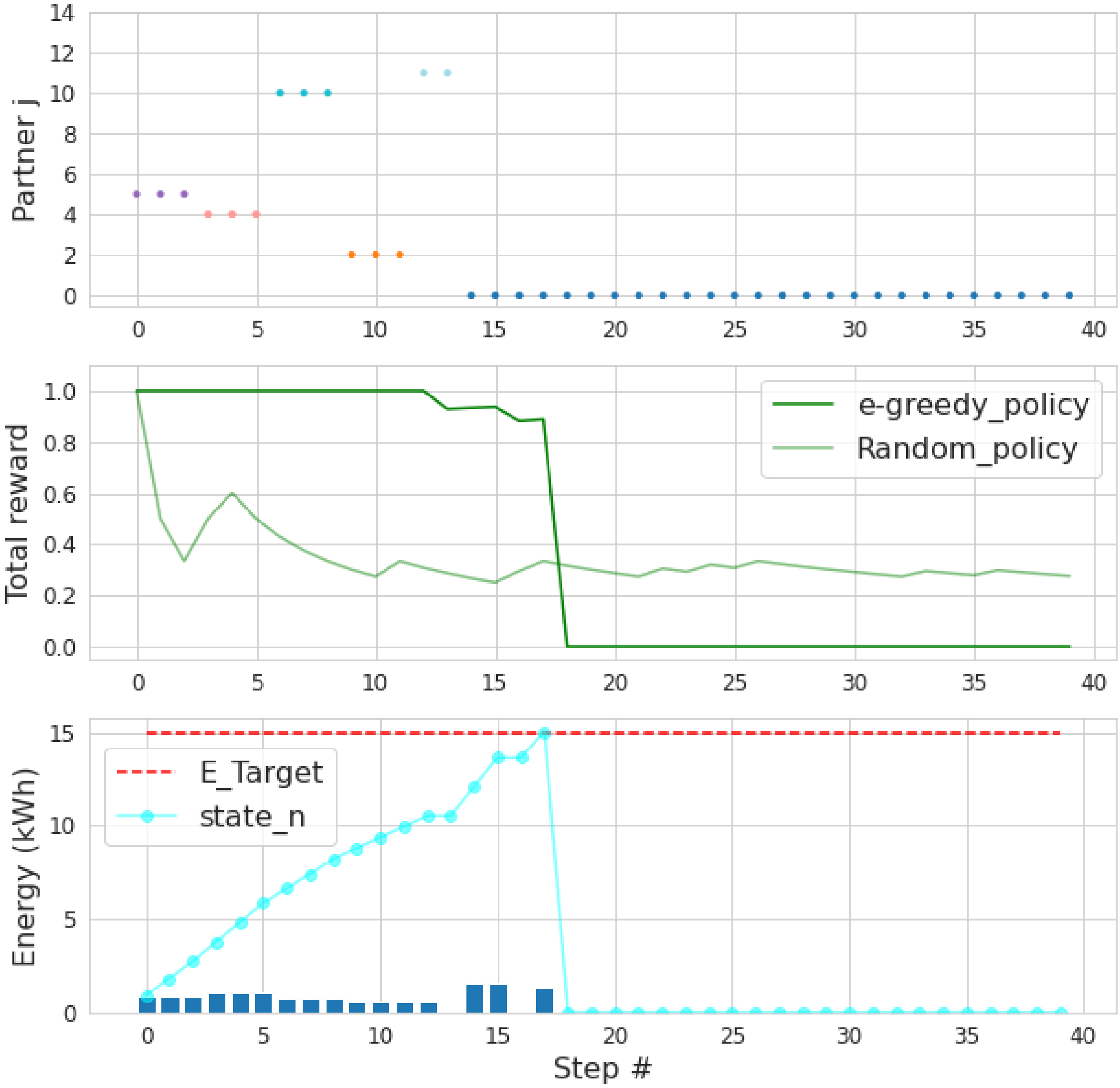
Compare strategies



Exploitation

We have clear winner

Epsilon-Greedy algorithm



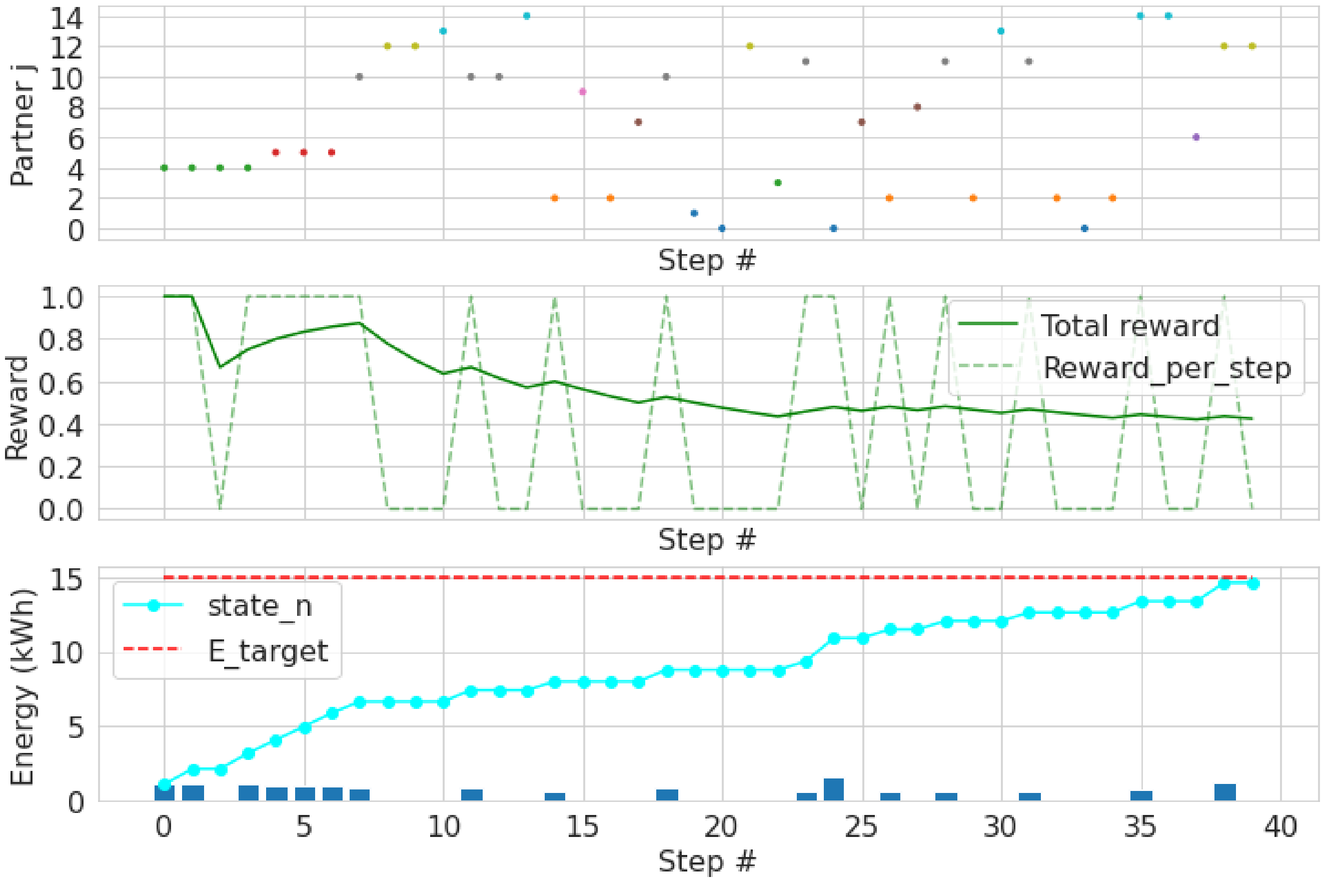
- Solution found on episode 91
  - $R_{Total} = 0.89$

- However, there is **no guarantee** to reach always this reward value:

$$\overline{R}_{Total} = 0.67 \qquad epi \in [21, 100]$$

Thompson Sampler algorithm

- We can also have 'bad' results even in the **exploitation phase**

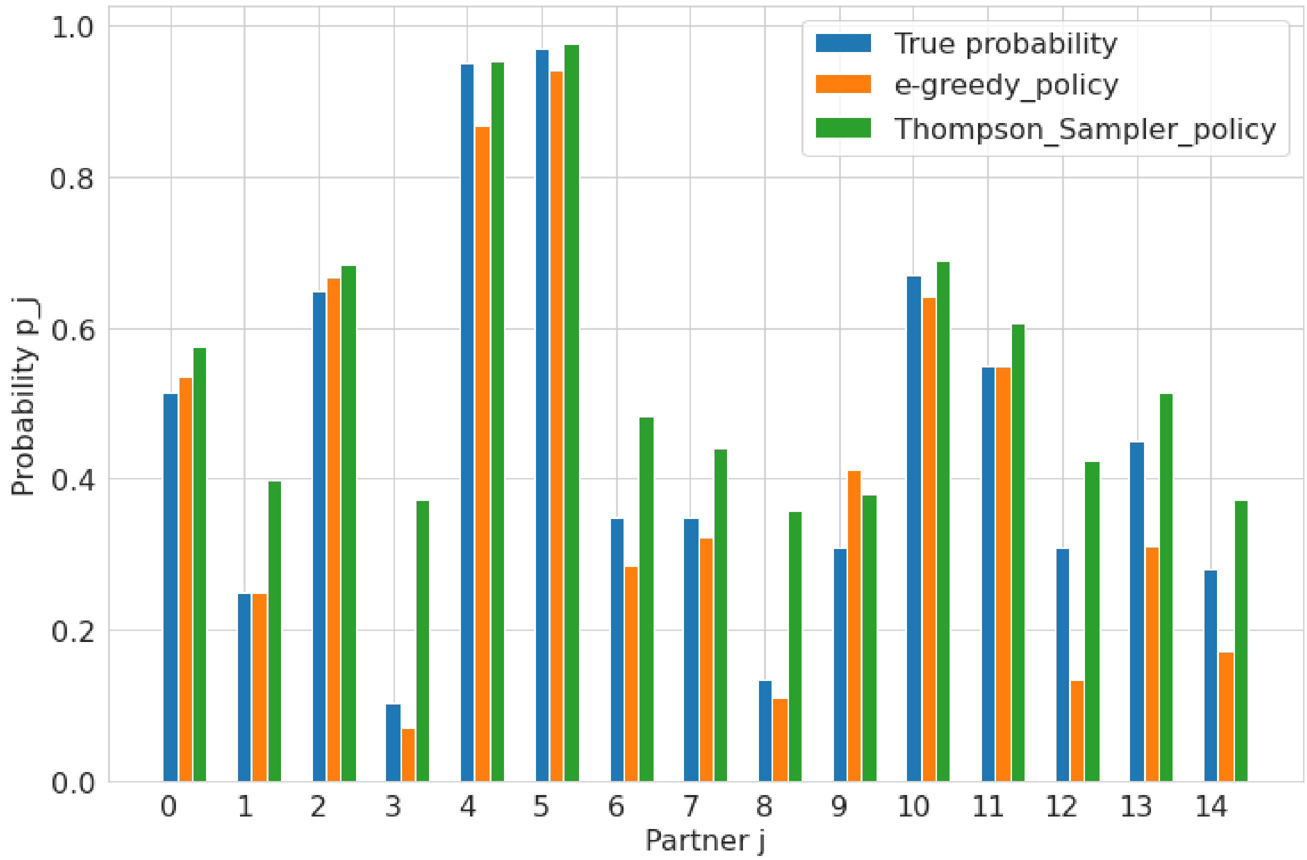


- Solution found on episode 91

$$R_{Total} = 0.41$$

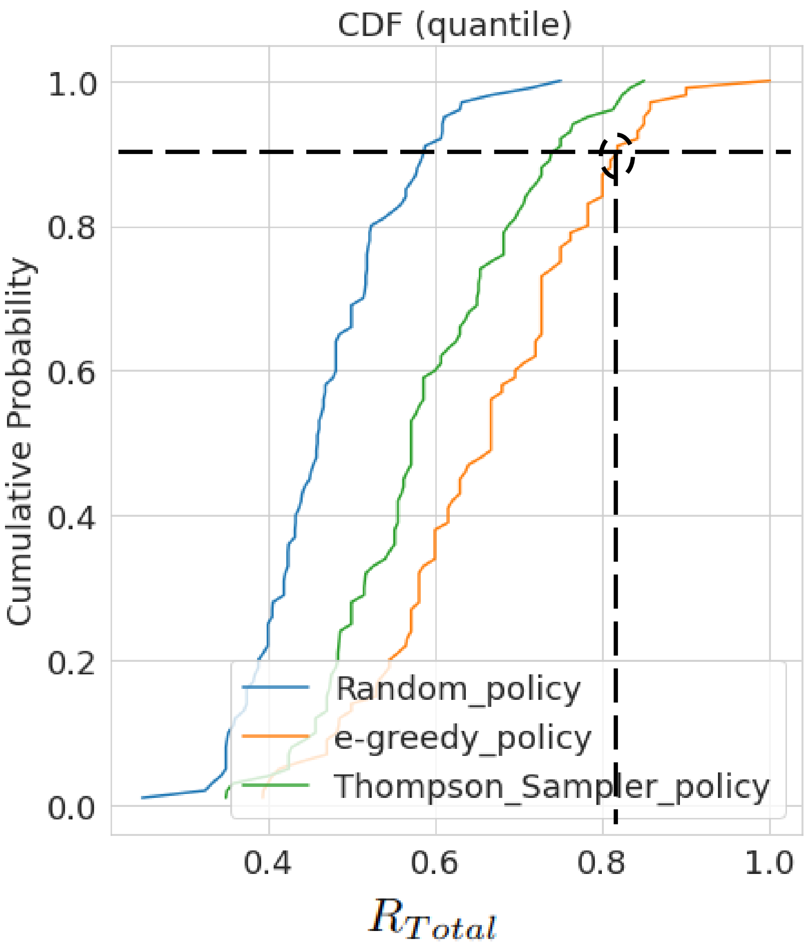
Optimal solution

- We can compute the estimator Q_j^*
  - $Q_j^* \approx p_j$
  - Calculate the mean Q_j^* for the last 10 episodes



Alternative approach

- In fact, we can retrieve the CDF of the R_{Total} for the 100 episodes



90% Percentile

- Filter the best episodes
- We can define an upper bound:

$$R_{Total} \geq 0.82$$

- We would then compute the estimator:
  - $Q_j^* \approx p_j$

Conclusions and next steps

- We are able to learn the partners with high success probability $Q_j^* \approx p_j$
- Easy way to build a learning agent via the Q-value (Action-value) functions
- Code available in my GitHub repo (link here)

- **Next steps:**
  - Estimate the Q_j^* using the CDF of the R_{Total}
  - Improve the Experience replay for the propagation
  - Assess the performance via a validation phase

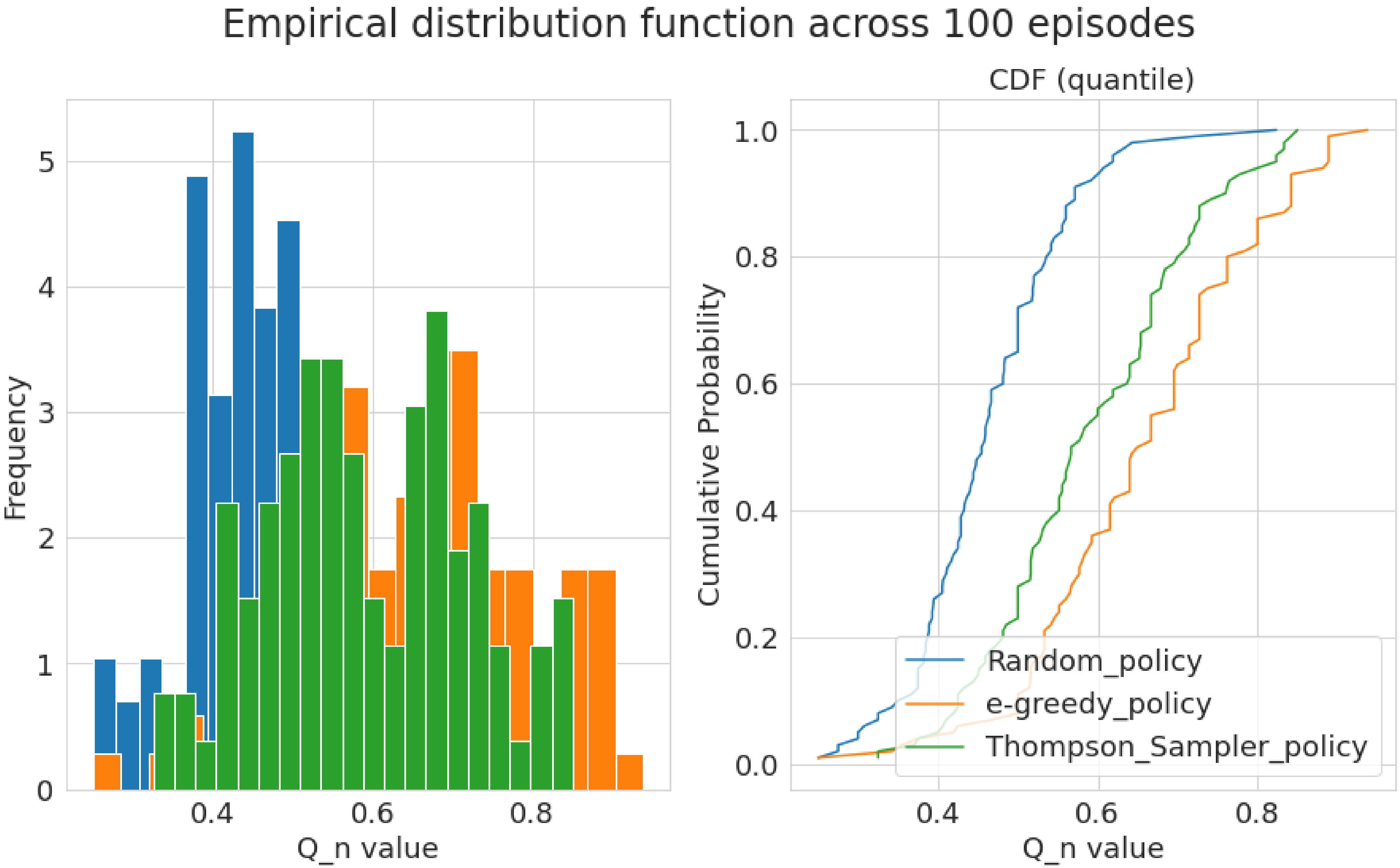
Danske Bank

Thanks for your attention!

Total reward performance

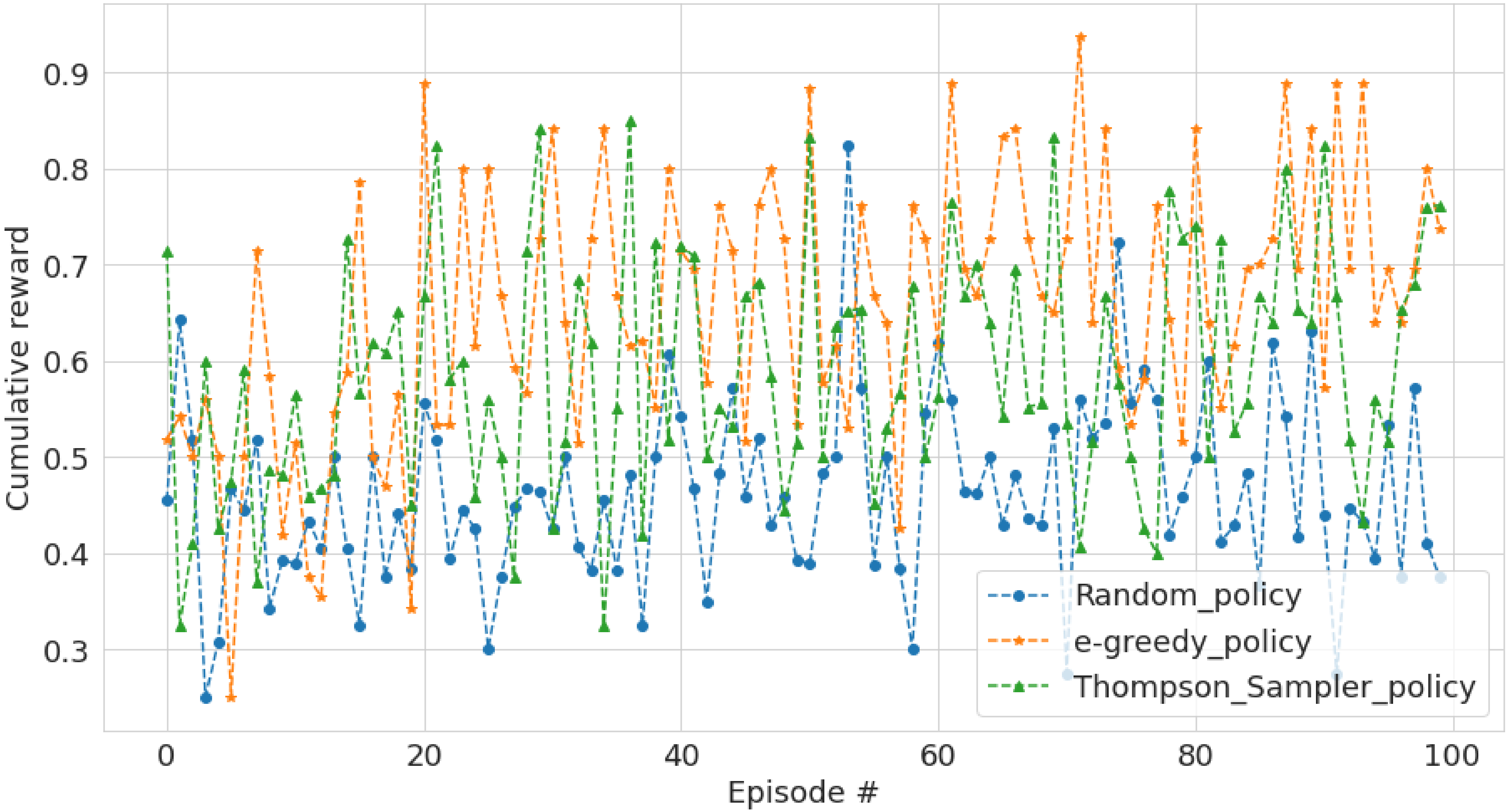
Algorithm	$\overline{R}_{Total}$ $epi \in [1, 20]$	$\overline{R}_{Total}$ $epi \in [21, 100]$
Random	0.49	0.47
ϵ-greedy	0.50	0.67

Danske Bank



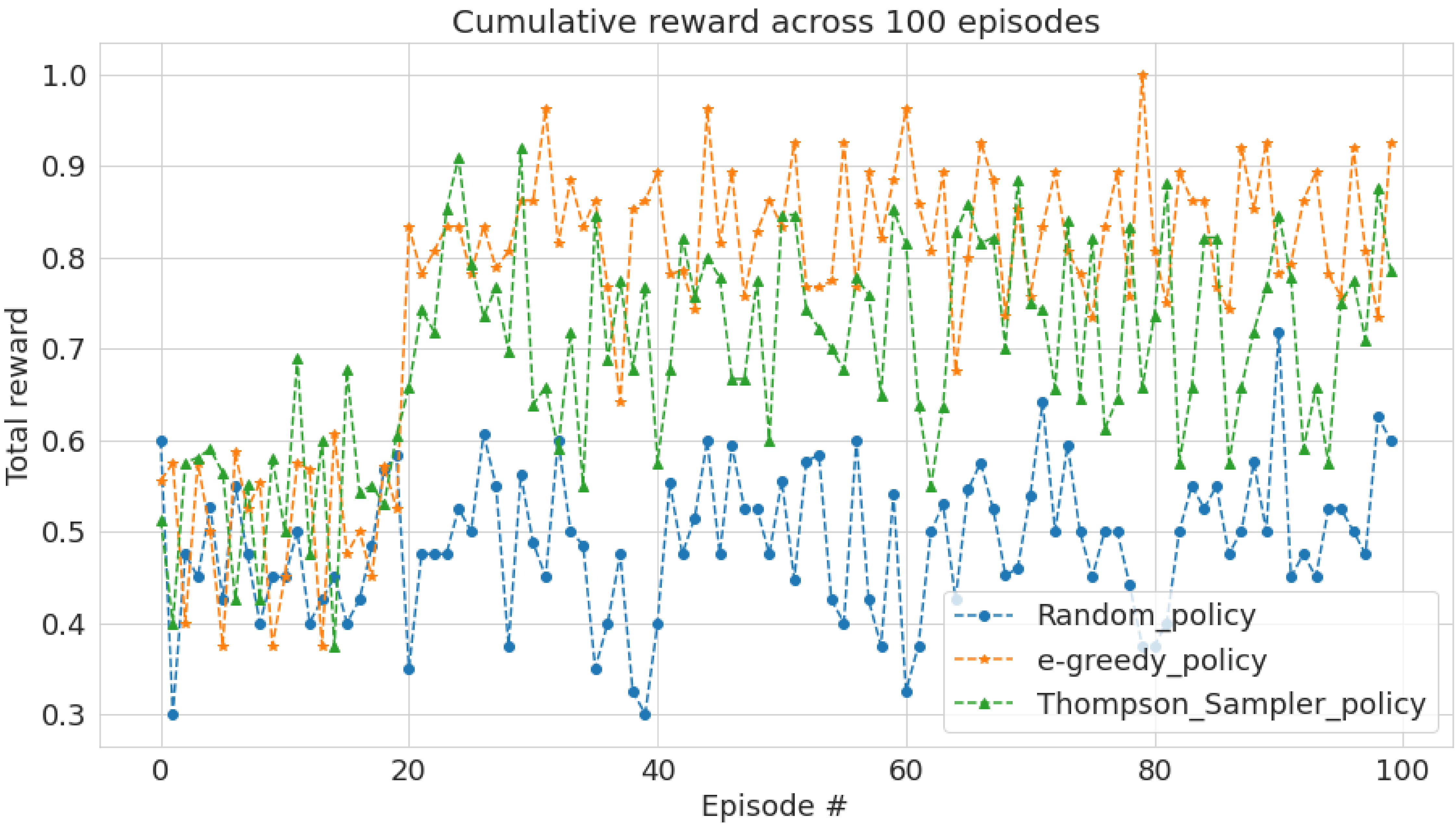
Empirical distribution function across 100 episodes
Frequency
Q_n value
CDF (quantile)
Cumulative Probability
Q_n value
Random_policy
e-greedy_policy
Thompson_Sampler_policy

Danske Bank



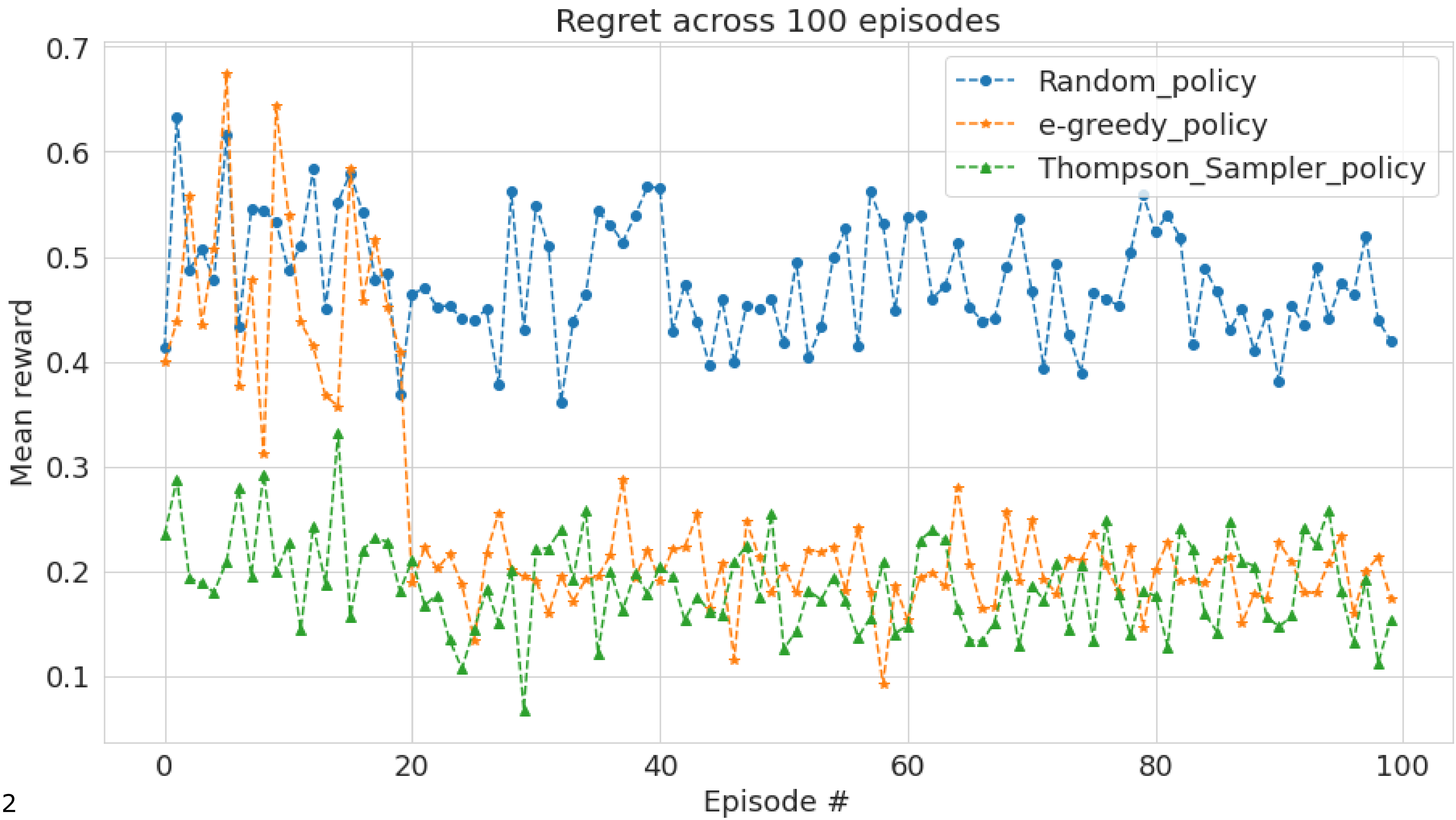
0.9
0.8
0.7
0.6
0.5
0.4
0.3
Cumulative reward
0
20
40
60
80
100
Episode #
Random_policy
e-greedy_policy
Thompson_Sampler_policy

Danske Bank

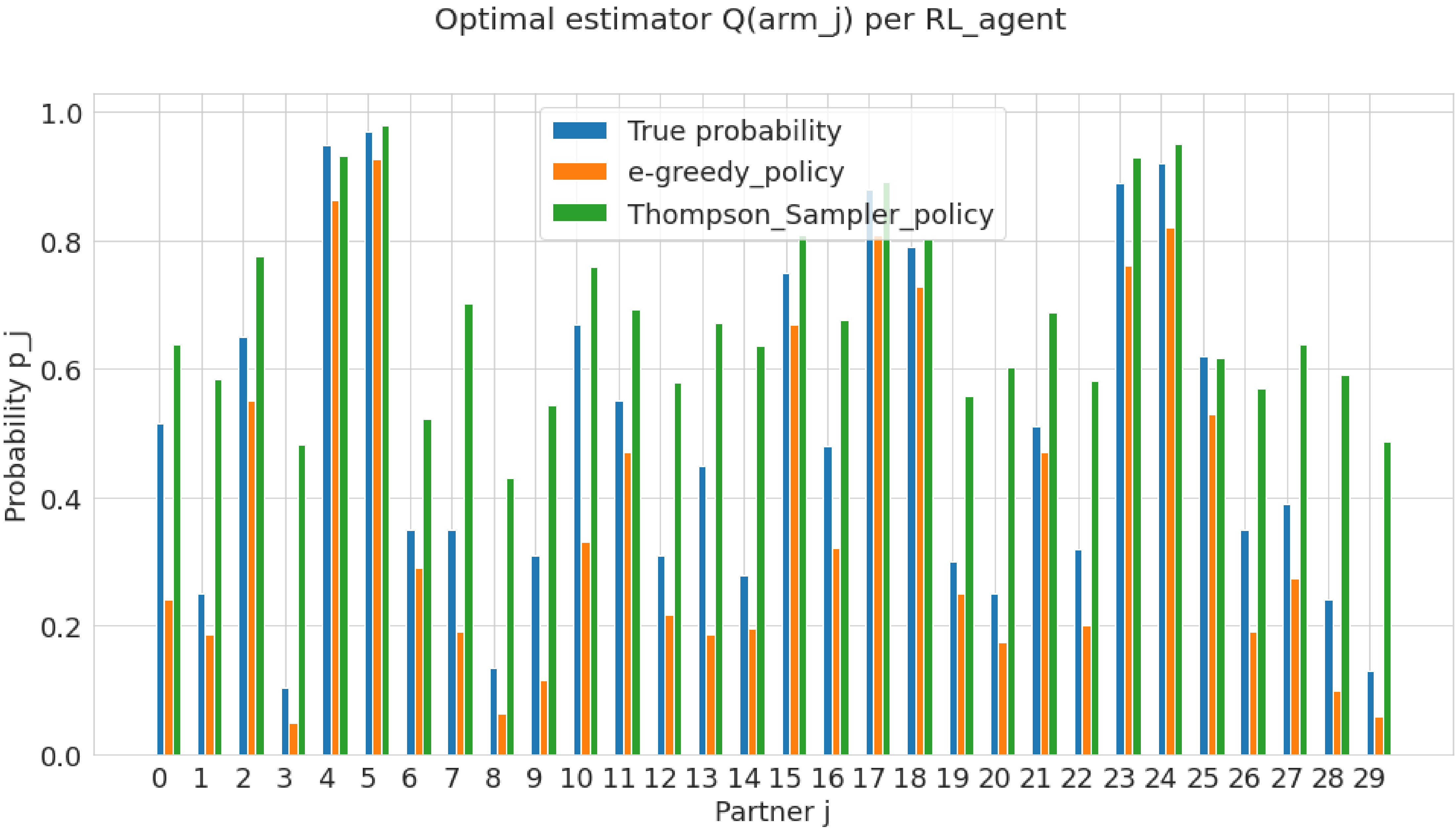


Cumulative reward across 100 episodes
Total reward
Episode #
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
20
40
60
80
100
Random_policy
e-greedy_policy
Thompson_Sampler_policy

Danske Bank



Regret across 100 episodes
Mean reward
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
20
40
60
80
100
Episode #
Random_policy
e-greedy_policy
Thompson_Sampler_policy


Optimal estimator Q(arm_j) per RL_agent
True probability
e-greedy_policy
Thompson_Sampler_policy
Probability p_j
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
Partner j