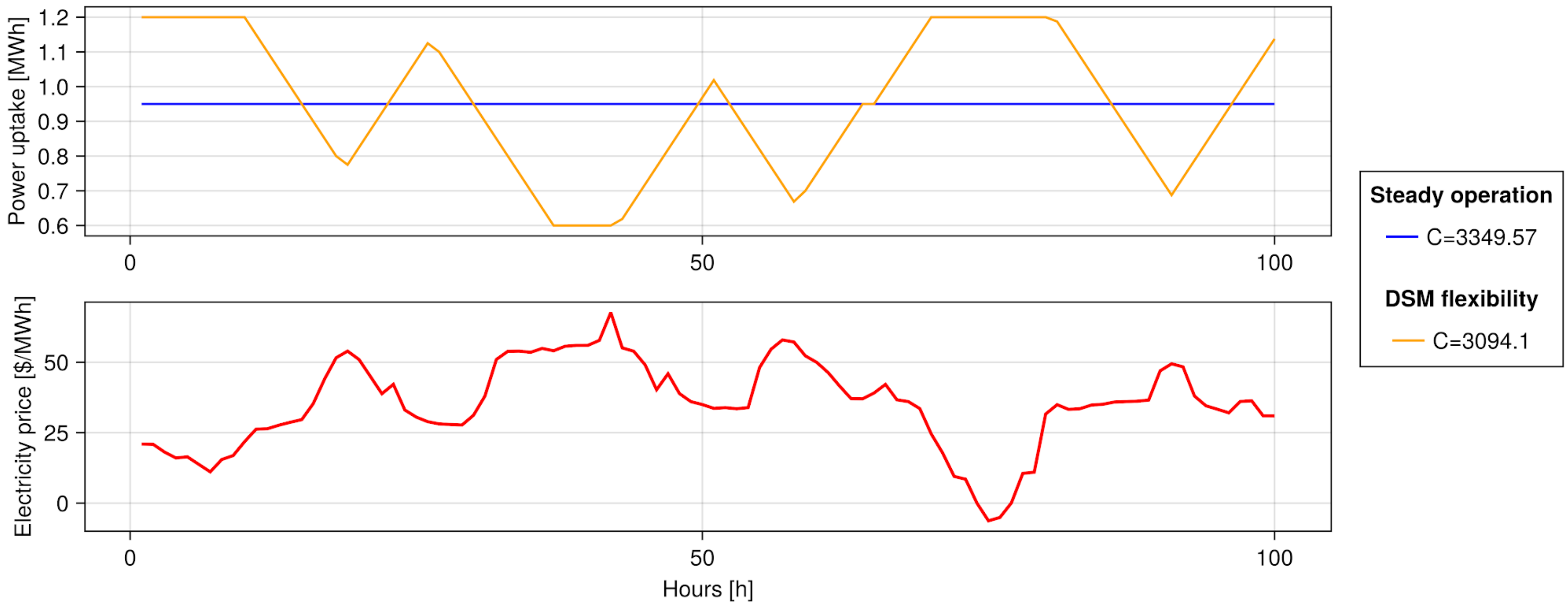
Deep Reinforcement Learning for Sequential Decision Making: A Case Study in Energy Management

Zhe Li, Amin Nassaji, Alexandra Janey

April 22, 2025

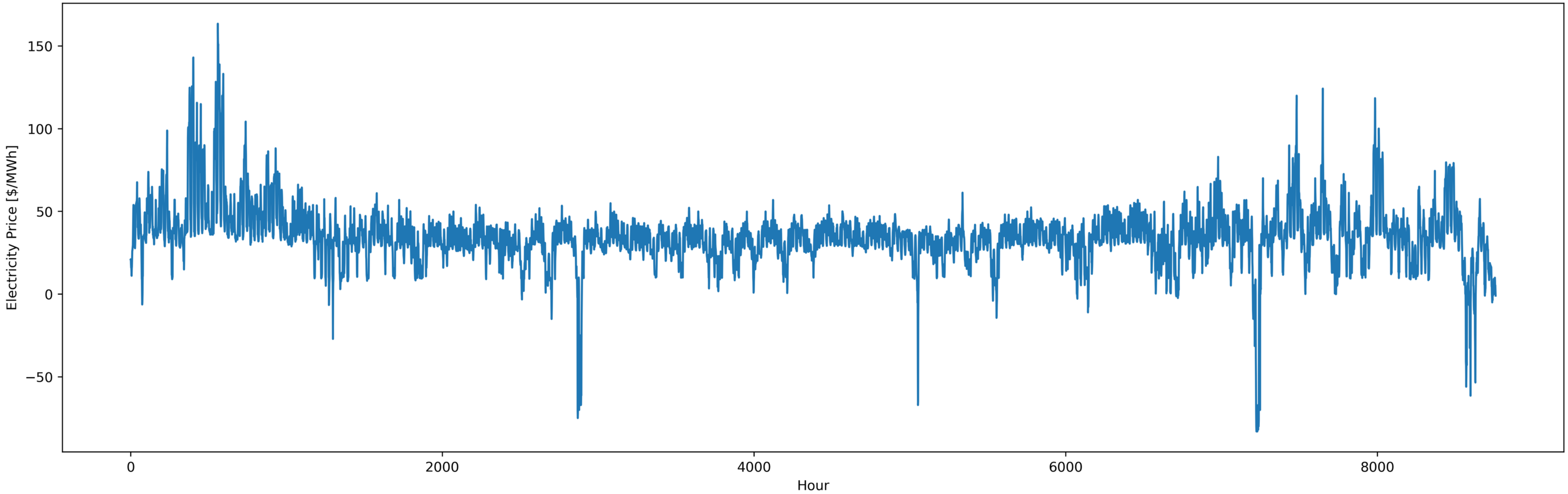
Problem Introduction

Electricity prices are volatile, so industrial demand side management (DSM) is a flexibility measure used for cost savings



Dataset

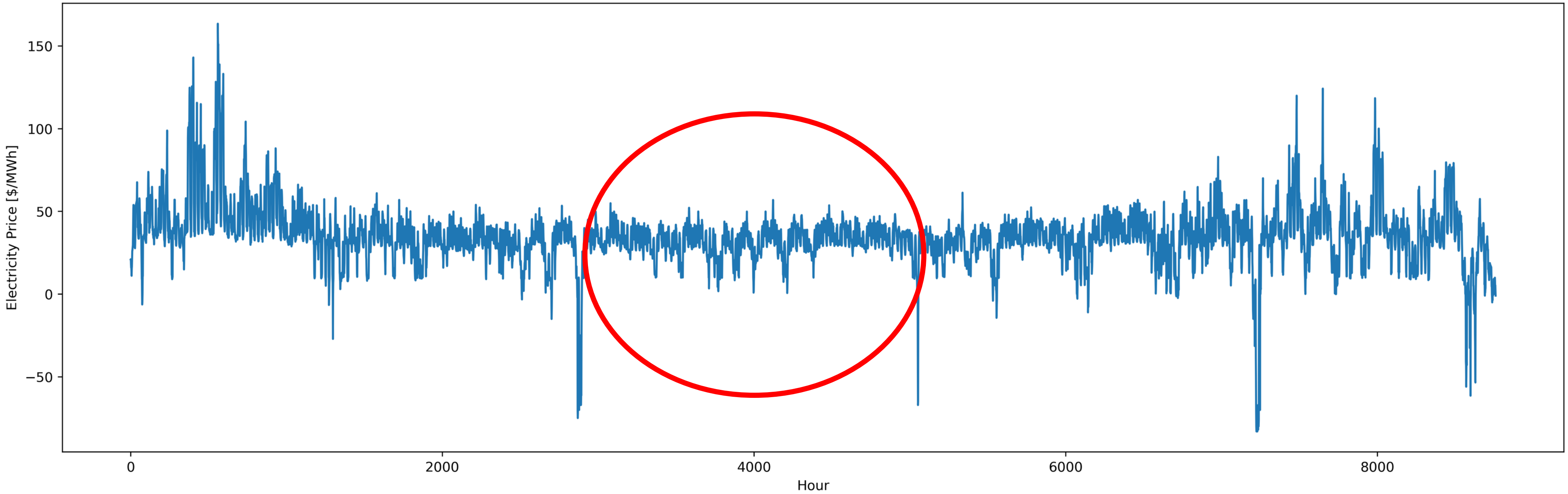
German electricity sector, hourly spot prices in 2017



Source: https://www.agora-energiewende.de/

Dataset

German electricity sector, hourly spot prices in 2017



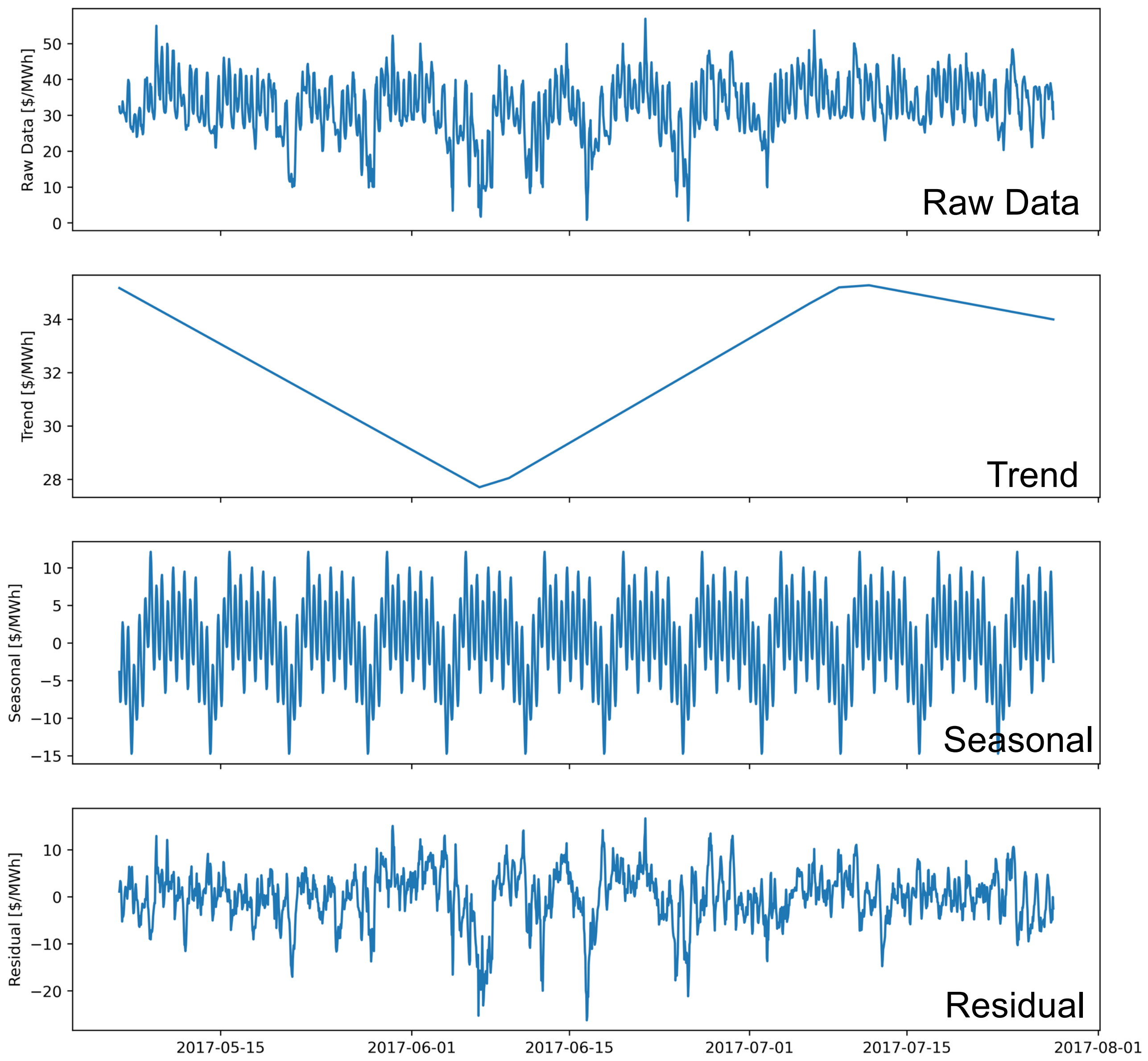
Source: https://www.agora-energiewende.de/

Time Series Analysis

Training data from

hours 3000 – 5000

(~May 6, 2017 – July 28, 2017)



Note: for test data we utilize

hours 5880 - 6048

Problem Description

- Traditional optimization methods have large computational costs when trying to implement in real time
- Treat scheduling model as a battery where we can “charge” it to store energy and use later
- Utilize sequential decision-making
- Solve this using Reinforcement Learning (RL) and compare it with the performance of Model Predictive Control (MPC)

Deterministic Model (Oracle)

Main variables: c_t (cost), P_t (power uptake), S_t (storage level), and Δ_t (ramping rate)

minimize $C = \sum_{t \in \mathcal{N}} c_t \cdot P_t$ — Minimize overall cost

subject to

$S_{t+1} = S_t + (P_t - P^{nom}) \quad \forall t \in \mathcal{N}$ — Storage level

$\Delta_{t+1} = P_{t+1} - P_t \quad \forall t \in \mathcal{N}$ — Ramping rate

$P^{nom} \cdot |\mathcal{N}| = \sum_{t \in \mathcal{N}} P_t$ — Electricity demand

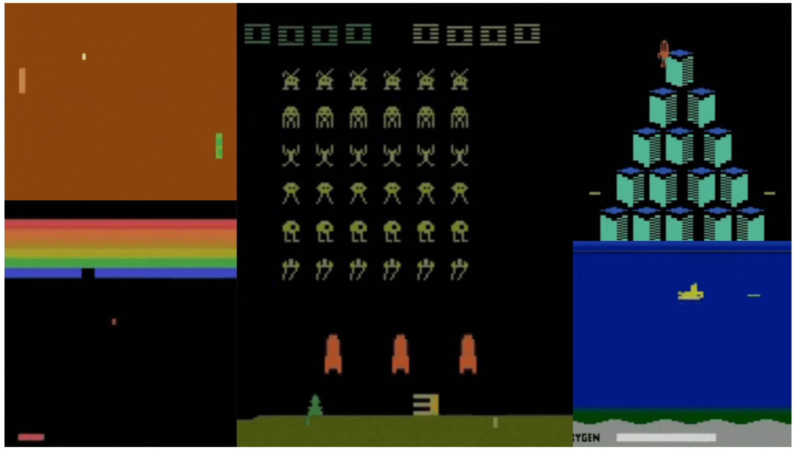
$P^{\min} \leq P_t \leq P^{\max} \quad \forall t \in \mathcal{N}$

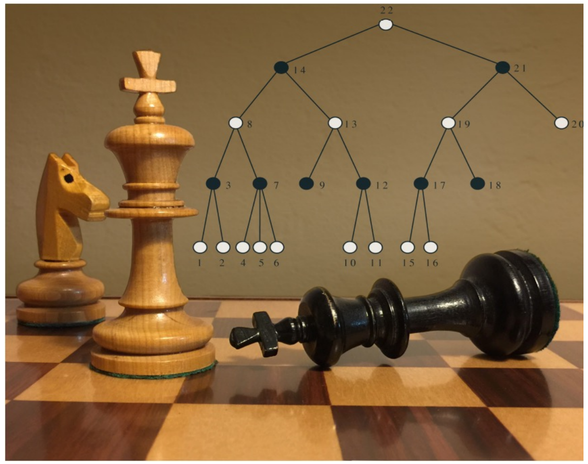
$-S^{max} \leq S_t \leq S^{max} \quad \forall t \in \mathcal{N}$

$-\Delta^{max} \leq \Delta_t \leq \Delta^{max} \quad \forall t \in \mathcal{N}$

Deep Reinforcement Learning: Overview

- What is RL?
  - Framework for solving sequential decision making problem
- How does it learn?
  - Trial and error from a world that provides occasional rewards
- What is the “Deep” part?
  - RL + neural network
- Applications
  - Games: Atari, Chess, Go…
  - Training large language model
  - Robotics controller



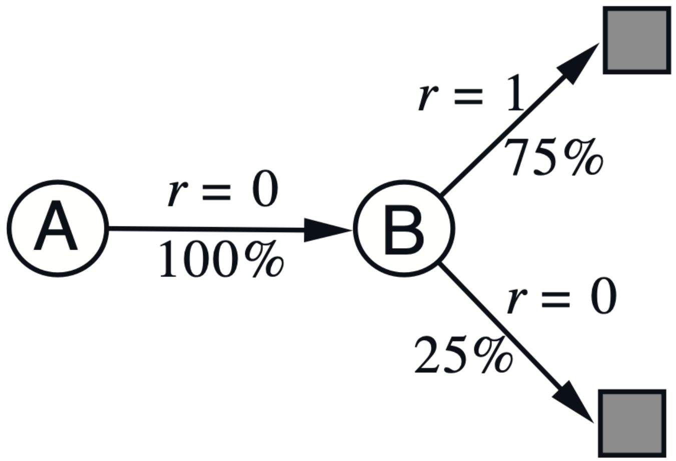


Framework: Markov Decision Process (MDP)

- Sequential decision problem:

$$(s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_t, a_t, r_t, s_{t+1}, \dots)$$

- s_t: state variable (<u>what we know</u>)
- a_t: action variable (<u>what we do</u>)
- $c_t = c(s_t, a_t)$: cost (<u>what we get</u>)
- $s' \sim p(s'|s, a)$: transition dynamics
- $a_t \sim \pi(s_t)$: policy (decision guidance)



Markov Decision Process Modeling

- State variables (what we know)

$$s_t = (S_t, P_{t-1}, c_t, \Delta c_t)$$

  - S_t: Storage level at time t
  - P_{t-1}: Power uptake at time t-1
  - c_t: Electricity price at time t
  - $\Delta c_t = c_t - c_{t-1}$: Price change from time t-1 to t

- Action variables (what we do)

$$a_t = (P_t)$$

  - $P_t \in [0.6, 1.2]$: Power uptake at time t

Markov Decision Process Modeling

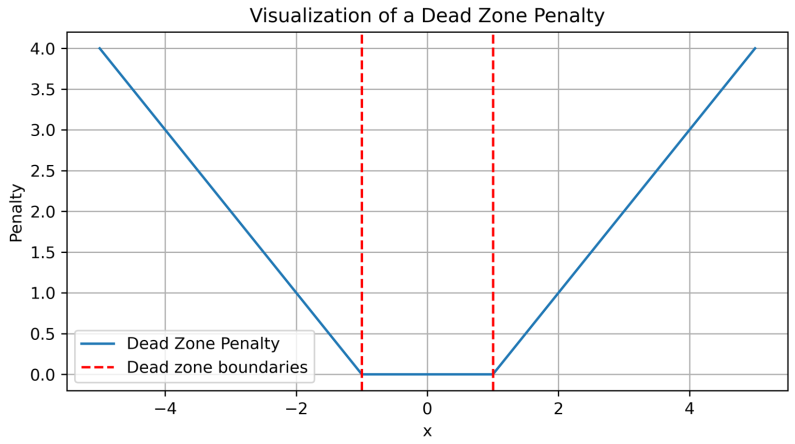
- Cost (what we get)

$$C_t = C(s_t, a_t) = c_t \cdot P_t + Penalty_{ramp} + Penalty_{storage}$$

  - $c_t \cdot P_t$: cost for purchasing electricity
  - $Penalty_{ramp}(\Delta_t)$: penalty for breaking ramping limit
  - $Penalty_{storage}(S_t, P_t)$: penalty for storage capacity

- Objective (cumulative cost):

$$\text{minimize } J = \sum_{t=0}^{N} C_t$$



Markov Decision Process Modeling

- Transition dynamics (how environment evolves)
  - Storage level: $S_{t+1} \leftarrow S_t + (P_t - P^{nom})$
  - Spot electricity price: $c_{t+1} \leftarrow Historical\ Data$
- Policy (what to do given the state)
  - Stochastic: $a_t \sim \pi(a|s)$
  - Deterministic: $a_t = \pi(s)$
- Task: Learn a good policy that minimizes cumulative cost

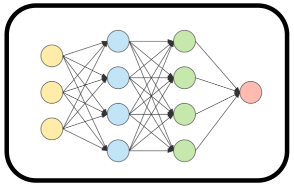
  w/ Reinforcement Learning

Deep Reinforcement Learning

Value-based RL

Learn a value function to evaluate action

s
a



$Q_\phi(s,a)$

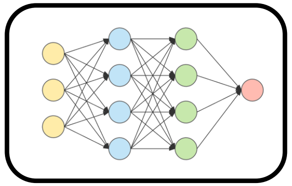
$$a^* = \underset{a}{\operatorname{argmax}}\, Q_\phi(s,a)$$

- Example: Q-learning
- Discrete action space

Policy-based RL

Learn a direct mapping from state to action

s



a^*

$$a^* = \pi_\theta\,(s)$$

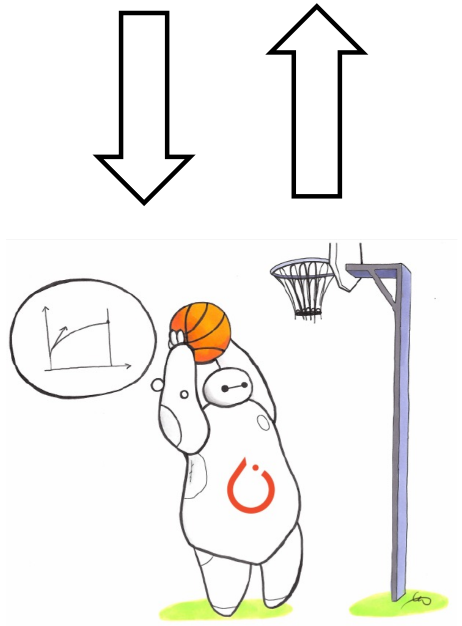
- Example: Proximal Policy Opt.
- Continuous action space

Reminder: $s_t = (S_t, P_{t-1}, c_t, \Delta c_t)$ $a_t = (P_t)$

Implementation Details

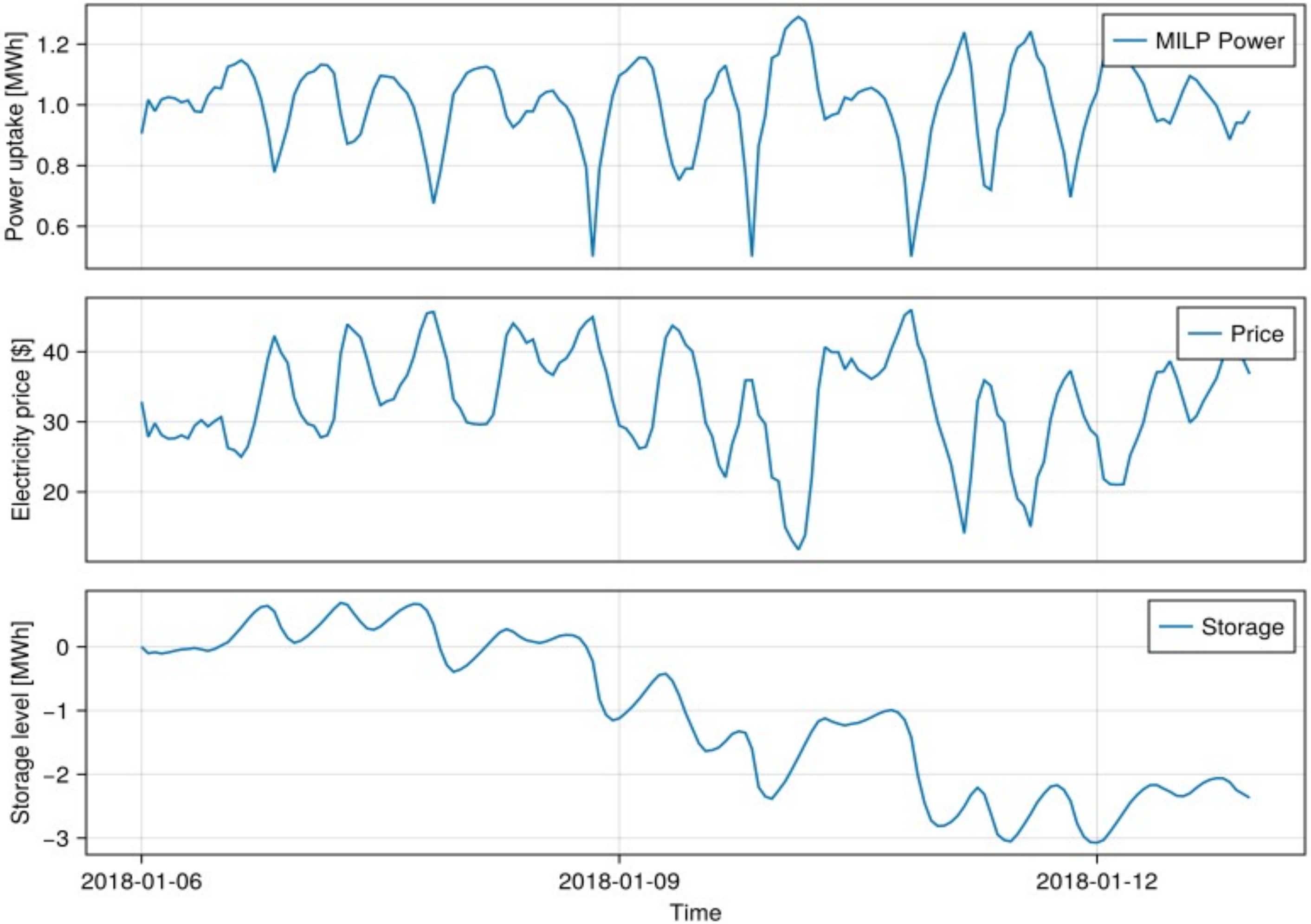
- Environment:
  - OpenAI’s Gymnasium library
- Training Algorithm:
  - PPO from Stable-Baselines3
- Hyperparameters:
  - Policy network: 'MlpPolicy'
  - Gamma: 0.995 (weight on long-term effect)
  - Learning rate: 3e-4

Gymnasium



Stable-Baselines3

Simulation results: MILP

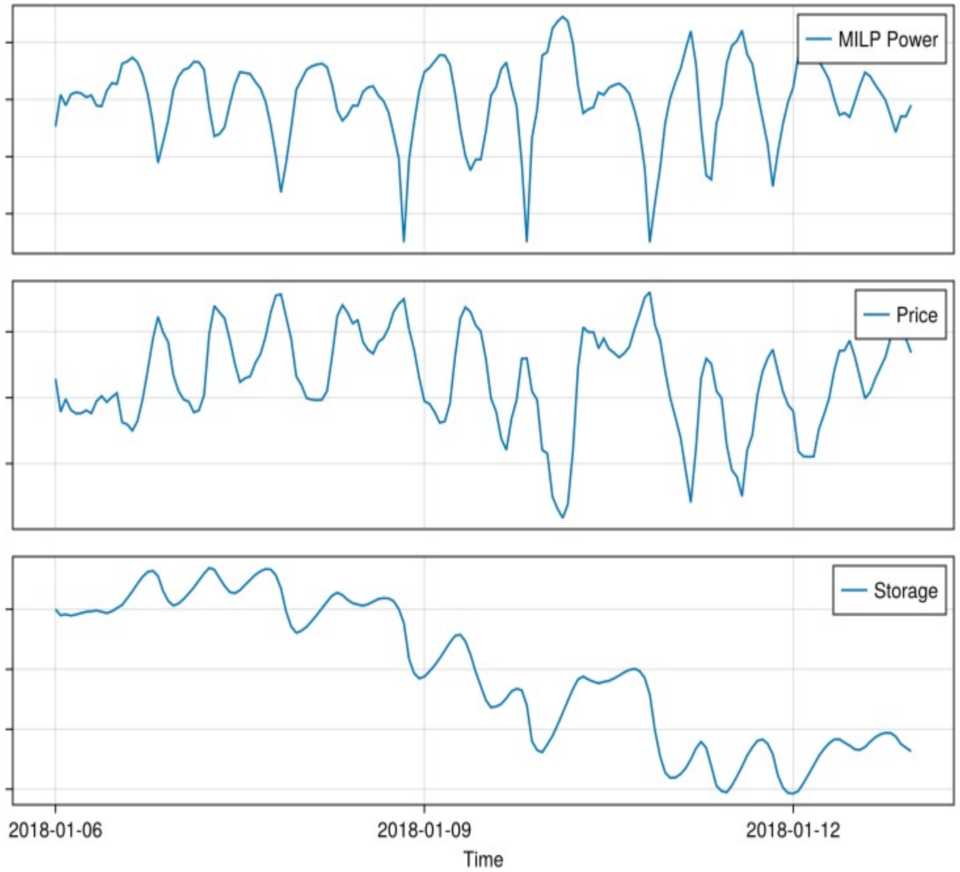


Total Cost:
$5273.82

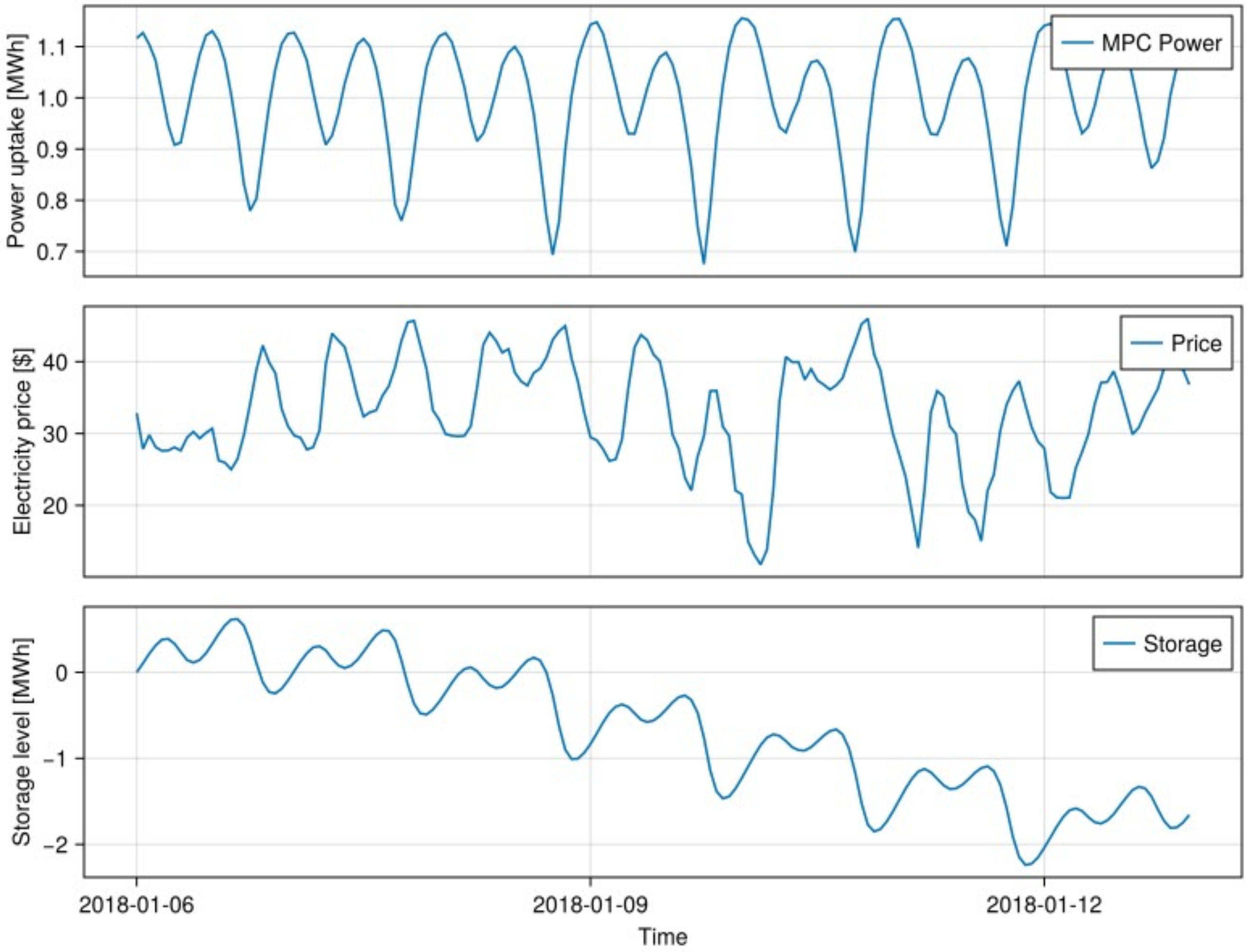
MILP: Strengths and Trade-off

- Pros
  - Global Optimality
  - Precise Constraint Handling
  - Efficient for Planning

- Cons
  - Modeling Nonlinearity
  - Requires Forecasts
  - Binary Variables0
  - Rigid Once Solved



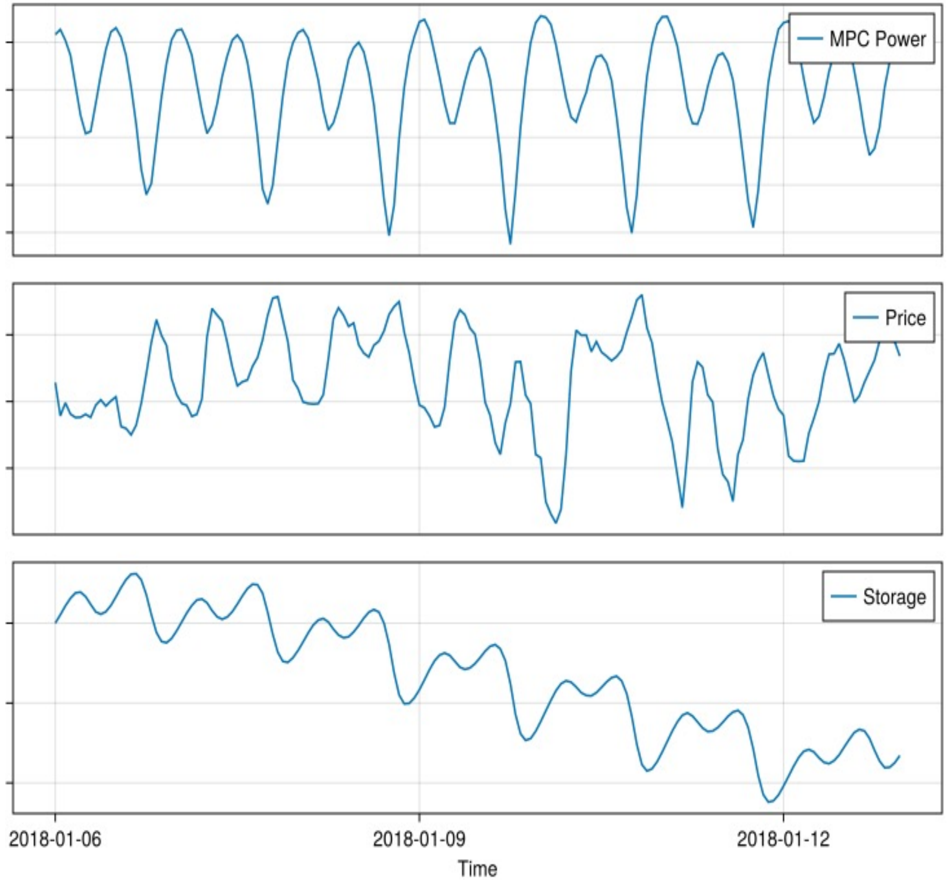
Simulation results: MPC



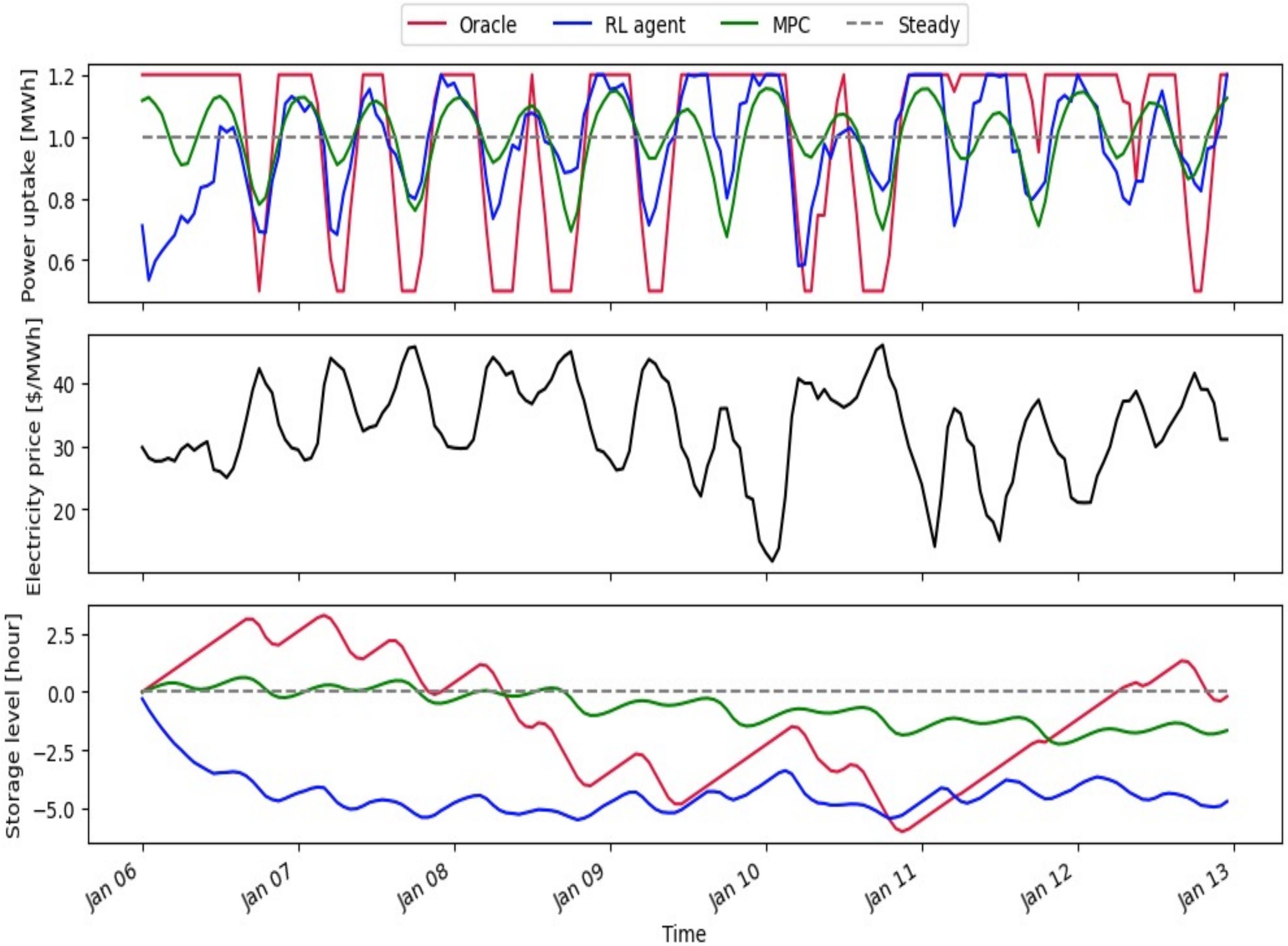
Total Cost:
$5462.73

MPC: Strengths and Trade-off

- Pros
  - Real-Time Adaptivity
  - Handles Constraints Naturally
  - Smooth Operation
  - Robust to Forecast Horizon
- Cons
  - Forecast Dependency
  - Computational Load
  - No Global Optimality

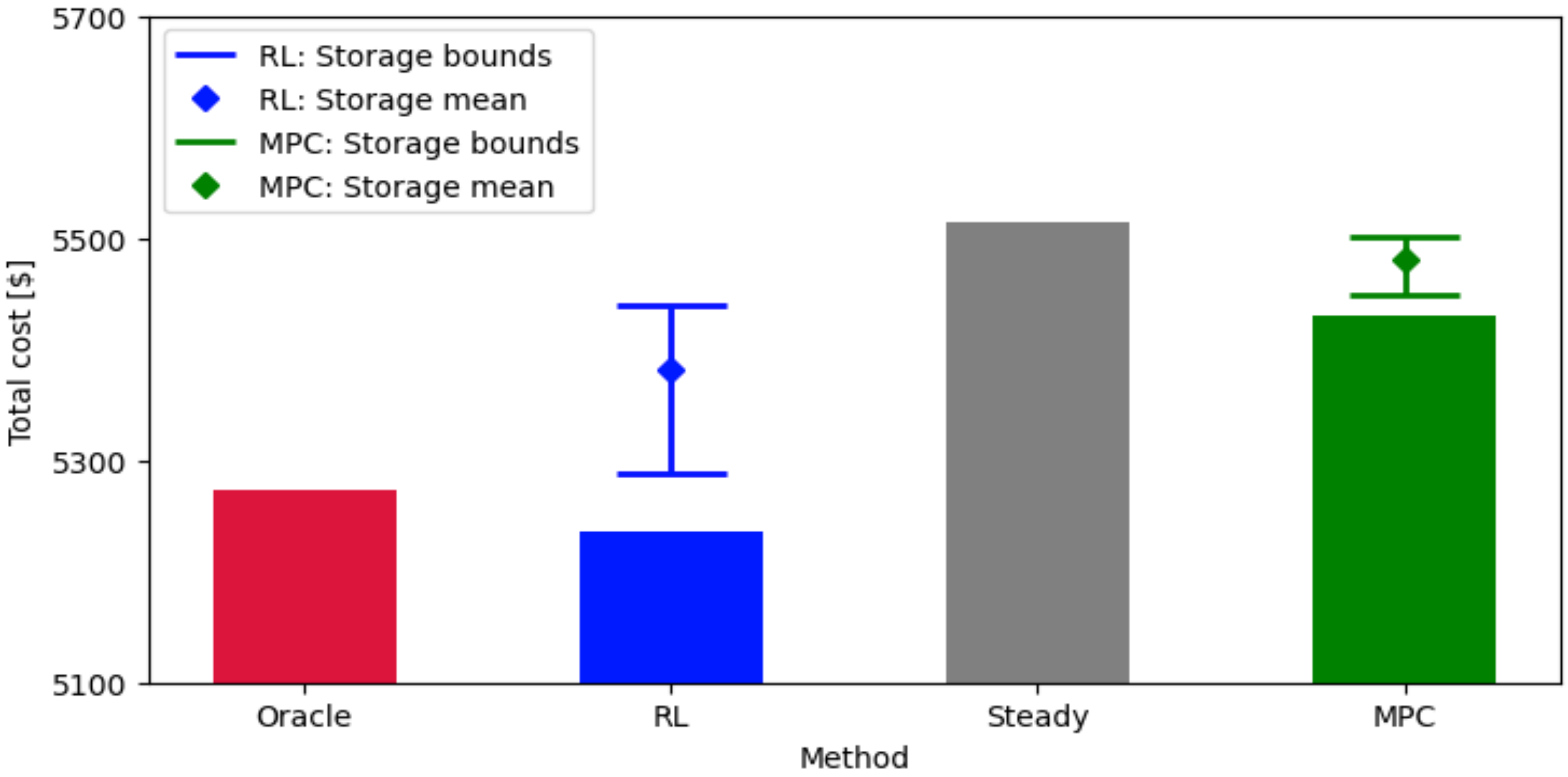


Simulation results: RL



Total Cost: $5236.00

Final Comparison and Conclusion



Final Comparison and Conclusion

- Pros
  - No Forecasting Required
  - Adaptivity
  - Smooth Behavior
  - Strong Cost Performance

- Cons
  - No Hard Guarantees
  - Training Time & Tuning
  - Less Transparent