

Figure 22: The performance of eSpark and MARL baselines in the MABIM 100 SKUs scenarios. eSpark(IPPO) refers to the configuration where the underlying MARL algorithm is IPPO, which is the setup discussed in the main text. eSpark(MAPPO) combines the framework with the MAPPO algorithm.  $\circ$  and  $\triangle$  indicate eSpark collects feedback here and regenerates the exploration function.

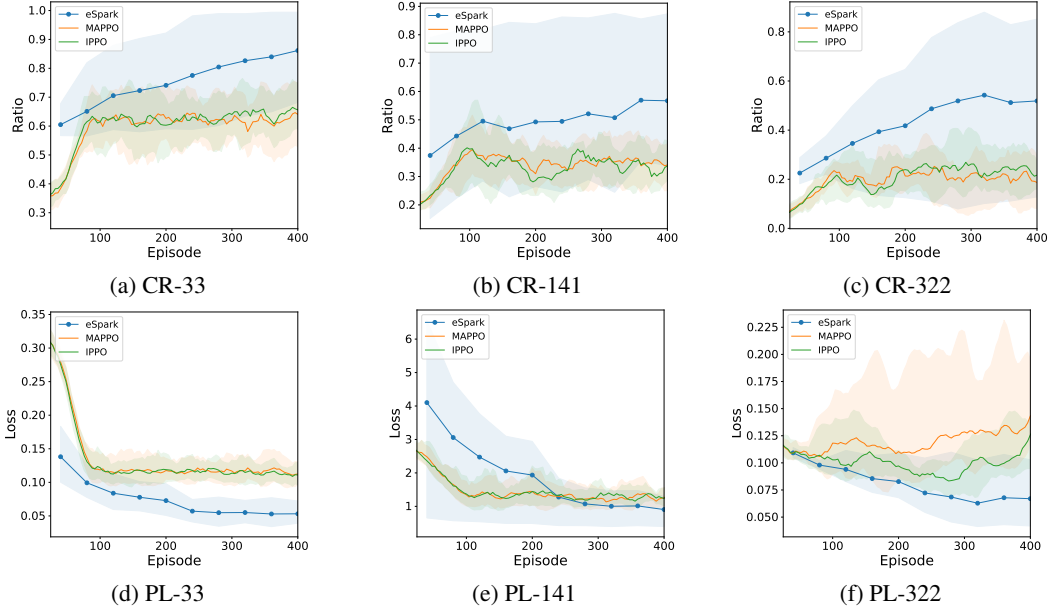


Figure 23: The performance of eSpark and MARL baselines in the active voltage control environment. A higher Control Rate (CR) and lower Power Loss (PL) indicate better algorithm performance. Our proposed eSpark outperforms all baselines and achieves the best results across all test scenarios.  $\circ$  indicates eSpark collects feedback here and regenerates the exploration function.

**GPT's input:**

Consider a resource allocation game with  $n$  agents. Each agent has a discrete action space  $\{0,1,2,3\}$ , representing the amount of resources allocated in each step. The goal is to maximize the team's cumulative utility while satisfying dynamic constraints that evolves over time.

**Game Rules:**

1. **Utility Function:** The team's cumulative utility is the sum of all agents' contributions, defined as:  $U(a_1, a_2, \dots, a_n) = \sum_{i=1}^n \log(1 + a_i)$
2. **Constraints:** At each time step  $t$ , a global constraint  $C_t$  limits the total allocation  $\sum_{i=1}^n a_i \leq C_t$
3. **Penalty for Violations:** If the total allocation exceeds  $C_t$ , a penalty proportional to the excess is subtracted from the utility:  $P(\text{excess}) = -k \cdot (\text{excess})^2$ , where  $k$  is a positive constant.

Please write an exploration function that masks invalid bids and bids likely to lead to inefficient budget usage (e.g., overbidding when the reward is low) for each agent. The function signature is:

```
def compute_mask(current_allocations, actions_space, constraints):
    return total_mask
```

current\_allocations: Current resource allocation of all agents.  
actions\_space: Available actions for each agent.  
constraints: The dynamic constraint  $C_t$ .

**GPT's output:**

- In this game, the exploration function must account for the following:
1. **Dynamic Constraints:** Predict whether an action will exceed  $C_t$  given the current state and other agents' possible actions.
  2. **Utility Contribution:** Estimate the marginal utility of each action and prioritize those with higher expected contributions.
  3. **Penalty Avoidance:** Mask actions likely to trigger penalties due to constraint violations.

```
import numpy as np
def compute_mask(current_allocations, actions_space, constraints):
    total_mask = []
    for action in actions_space:
        predicted_allocation = sum(current_allocations) + action

        # Check if the action violates the constraint
        if predicted_allocation > constraints:
            total_mask.append(0) # Mask the action
        else:
            # Estimate marginal utility and penalize low
            contribution
            marginal_utility = np.log(1 + action)
            penalty = -0.1 * (predicted_allocation - constraints)
            ** 2 if predicted_allocation > constraints else 0
            # Mask actions with low marginal utility or high
            penalties
            if marginal_utility + penalty < 0.5: # Threshold
                chosen heuristically
                total_mask.append(0)
            else:
                total_mask.append(1)
    return total_mask
```