ENGINEERING Information Technology & Electronic Engineering
https://www.academax.com/EITEE; https://www.engineering.org.cn
https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=11344065
E-mail: eng.itee@zju.edu.cn

Supplementary materials for

Zhongyang MAO, Jiahuan GENG, Faping LU, Wenbiao TIAN, Jiafang KANG, Yaozong PAN, 2026. An adaptive meta-reinforcement learning scheme for maritime dynamic spectrum access and service scheduling. *ENGINEERING Inform Technol Electron Eng*, 27(8):260047. https://doi.org/10.1631/ENG.ITEE.2026.0047

1 Complete PseudoCodes

Algorithm S1: MADDPG for Dynamic Spectrum Environment

Algorithm S1 MADDPG for Dynamic Spectrum Environment

Input:

- Task distribution $\mathcal{T}$ with parameters $\{\lambda_e, \lambda_n, \tau_e, \tau_n, \upsilon, \zeta\}$
- Learning rate α and discount factor γ
- Replay buffer size B, training episodes E, test episodes E_{test}

Output:

- Trained actor parameters θ^* and Performance metrics

Meta-Training Phase

Initialize actor θ_i and critic ϕ_i for each agent i
Initialize target networks $\theta'_i \leftarrow \theta_i$, $\phi'_i \leftarrow \phi_i$
Initialize replay buffer $\mathcal{D}$
for $episode = 1$ **to** E **do**
  Sample task $\mathcal{T}_j$ from $\mathcal{T}$
  Initialize environment with $\mathcal{T}_j$ parameters
  for each agent i **do**
    Select action $a_i \sim \pi_{\theta_i}(a_i|o_i)$ with exploration noise
  end for
  Execute joint action a, get reward r and next observations o'
  Store (o, a, r, o') in $\mathcal{D}$ and $o \leftarrow o'$
  Sample mini-batch from $\mathcal{D}$
  for each agent i **do**
    Compute target $y_i = r_i + \gamma \cdot Q_{\phi'_i}(o', a')$
    Update critic: $\mathcal{L}(\phi_i) = \mathbb{E}[(Q_{\phi_i}(o, a) - y_i)^2]$
    Update actor: $\nabla_{\theta_i} J(\theta_i) \approx \nabla_{\theta_i} \log \pi_{\theta_i}(a_i|o_i) \cdot \nabla_{a_i} Q_{\phi_i}(o, a)$
  end for
  Update target networks
end for

Meta-Testing Phase

Load trained parameters θ^*
for $episode = 1$ **to** E_{test} **do**
  Sample test task $\mathcal{T}_{test}$ from $\mathcal{T}$
  Initialize environment with $\mathcal{T}_{test}$ parameters
  for each agent i **do**
    Select action $a_i = \arg\max \pi_{\theta_i}(a_i|o_i)$
  end for
  Execute joint action a, get reward r and next observations o'
  Accumulate reward and $o \leftarrow o'$
end for
Compute average performance metrics

Algorithm S2: MetaSASS for Dynamic Spectrum Environment

Algorithm S2 MetaSASS for Dynamic Spectrum Environment

Input:

- Task distribution $\mathcal{T}$ with parameters $\{p_{00}, p_{10}, \lambda_e, \lambda_n, \tau_e, \tau_n, \upsilon, \zeta\}$
- Meta-learning rate β, inner-loop learning rate α
- Meta-batch size N, inner-loop steps T, meta-iterations E
- Adaptation steps T_{adapt}, test episodes E_{test}

Output:

- Trained meta-policy parameters θ^* and Performance metrics

Meta-Training Phase

Initialize meta-parameters θ
for $iter = 1$ **to** E **do**
  Sample meta-batch $\{\mathcal{T}_1, \ldots, \mathcal{T}_N\} \sim p(\mathcal{T})$
  for each task $\mathcal{T}_j$ in meta-batch **do**
    $\theta_i' \leftarrow \theta$
    for $step = 1$ **to** T **do**
      Collect trajectories τ_j using $\pi_{\theta_i'}$ in $\mathcal{T}_j$
      $\theta_i' \leftarrow \theta_i' - \alpha \cdot \nabla_{\theta_i'} \mathcal{L}_{\mathcal{T}_j}(\theta_i')$
    end for
    Accumulate meta-gradient: $\mathcal{L}_{meta} \leftarrow \mathcal{L}_{meta} + \mathcal{L}_{\mathcal{T}_j}^{val}(\theta_i')$
  end for
  Update meta-parameters: $\theta \leftarrow \theta - \beta \cdot \nabla_\theta \mathcal{L}_{meta}$
end for

Meta-Testing Phase

Load meta-parameters θ^*
for $episode = 1$ **to** E_{test} **do**
  Sample test task $\mathcal{T}_{test}$ from $\mathcal{T}$
  $\theta_{adapt} \leftarrow \theta^*$
  for $step = 1$ **to** T_{adapt} **do**
    Collect trajectories τ_{test} using $\pi_{\theta_{adapt}}$ in $\mathcal{T}_{test}$
    $\theta_{adapt} \leftarrow \theta_{adapt} - \alpha \cdot \nabla_{\theta_{adapt}} \mathcal{L}_{\mathcal{T}_{test}}(\theta_{adapt})$
  end for
  Evaluate performance using adapted policy $\pi_{\theta_{adapt}}$ in $\mathcal{T}_{test}$
end for
return θ^*, performance metrics

Algorithm S3: AMRLSASS for Dynamic Spectrum Environment

Algorithm S3 AMRLSASS for Dynamic Spectrum Environment

Input:

- Task distribution $\mathcal{T}$ with parameters $\{p_{00}, p_{01}, \lambda_e, \lambda_n, \tau_e, \tau_n, \upsilon, \zeta\}$
- Meta-learning rate β, base learning rate α_{base}, encoder learning rate α_{enc}
- Meta-batch size N, inner-loop steps T, meta-iterations E
- Adaptation steps T_{adapt}, test episodes E_{test}, decay factor η

Output:

- Trained meta-policy parameters θ^*, adaptive module parameters φ^*
- Performance metrics after adaptive adaptation

Meta-Training Phase

Initialize meta-parameters θ and adaptive module parameters φ
for $iter = 1$ **to** E **do**
  Sample meta-batch $\{\mathcal{T}_1, \ldots, \mathcal{T}_N\} \sim p(\mathcal{T})$
  for each task $\mathcal{T}_j$ in meta-batch **do**
    $\theta_i' \leftarrow \theta$, $\varphi' \leftarrow \varphi$
    for $step = 1$ **to** T **do**
      Collect trajectories τ_j in $\mathcal{T}_j$
      Extract task features from τ_j
      Compute task difficulty d_{task} using φ'
      Compute adaptive learning rate: $\alpha_{step} = \alpha_{base} \cdot (1.5 - d_{task}) \cdot (1 - \eta \cdot \frac{step}{T})$
      Compute gradient $\nabla_{\theta_i'} \mathcal{L}_{\mathcal{T}_j}(\theta_i')$
      Update meta-parameters: $\theta_i' \leftarrow \theta_i' - \alpha_{step} \cdot \nabla_{\theta_i'} \mathcal{L}_{\mathcal{T}_j}(\theta_i')$
      Update adaptive module parameters: $\varphi' \leftarrow \varphi' - \alpha_{enc} \cdot \nabla_{\varphi'} \mathcal{L}_{enc}(\varphi')$
    end for
    Accumulate meta-gradient: $\mathcal{L}_{meta} \leftarrow \mathcal{L}_{meta} + \mathcal{L}_{\mathcal{T}_j}^{val}(\theta_i')$
  end for
  Update meta-parameters: $\theta \leftarrow \theta - \beta \cdot \nabla_\theta \mathcal{L}_{meta}$
  Update adaptive module parameters: $\varphi \leftarrow \varphi - \beta_{enc} \cdot \nabla_\varphi \mathcal{L}_{meta}^{enc}$
end for

Meta-Testing Phase

Load meta-parameters θ^* and adaptive module parameters φ^*
for $episode = 1$ **to** E_{test} **do**
  Sample test task $\mathcal{T}_{test}$ from $\mathcal{T}$
  $\theta_{adapt} \leftarrow \theta^*$, $\varphi_{adapt} \leftarrow \varphi^*$
  for $step = 1$ **to** T_{adapt} **do**
    Collect trajectories τ_{test} using $\pi_{\theta_{adapt}}$ in $\mathcal{T}_{test}$
    Compute task difficulty d_{task}
    Compute adaptive learning rate: $\alpha_{adapt} = \alpha_{base} \cdot (1.5 - d_{task})$
    Update adapted parameters: $\theta_{adapt} \leftarrow \theta_{adapt} - \alpha_{adapt} \cdot \nabla_{\theta_{adapt}} \mathcal{L}_{\mathcal{T}_{test}}(\theta_{adapt})$
  end for
  Evaluate performance using adapted policy $\pi_{\theta_{adapt}}$ in $\mathcal{T}_{test}$
end for
return θ^*, performance metrics

2 Additional Tables

Table S1 Complete hyperparameters

Parameter	Value	Description
K	20	Total number of channels
p_{00}	(0,1)	Channel state transition probability from idle to idle
p_{10}	(0,1)	Channel state transition probability from occupied to idle
τ	0.01	Target network soft update coefficient
L	6	Number of channels sensed per SU
γ	0.95	Discount factor
λ_e	[0.3,1.5]	Urgent packet arrival rate
λ_n	[0.8,3.0]	Normal packet arrival rate
τ_e	[4,12]	Urgent packet time-to-live
τ_n	[7,20]	Normal packet time-to-live
υ	[6,20]	Hopping sequence
ς	[2,6]	Spreading factor
Seed	42	Random seed
α_{actor}	0.0001	Actor network learning rate
α_{critic}	0.0005	Critic network learning rate
α_{adapt}	0.001	Adaptation learning rate
β	0.001	Meta learning rate
E	10	Meta-learning inner-loop steps
$\mathcal{D}$	20	Experience samples per task
ω_{emerg_succ}	3	Reward for successful Urgent packet schedule
ω_{normal_succ}	1	Reward for successful Normal packet schedule
$\omega_{emerg\_timeout}$	-3	Penalty for Urgent packet timeout
$\omega_{normal\_timeout}$	-1	Penalty for Normal packet timeout
ω_{coll}	-0.5	Penalty for channel collision

Table S2 Training task parameters

Parameter	Value	Parameter	Value
p_{00}	[0.25, 0.35]	τ_e	[4, 6]
p_{10}	[0.15, 0.25]	τ_n	[7, 9]
λ_e	[0.3, 0.6]	υ	[6, 10]
λ_n	[0.8, 1.2]	ς	[2, 3]

Table S3 Testing task parameters

Parameter	Value	Parameter	Value
p_{00}	[0.5, 0.8]	τ_e	[8, 12]
p_{10}	[0.05, 0.12]	τ_n	[14, 20]
λ_e	[0.8, 1.5]	υ	[13, 20]
λ_n	[1.8, 3.0]	ς	[4, 6]

3 Additional Figures

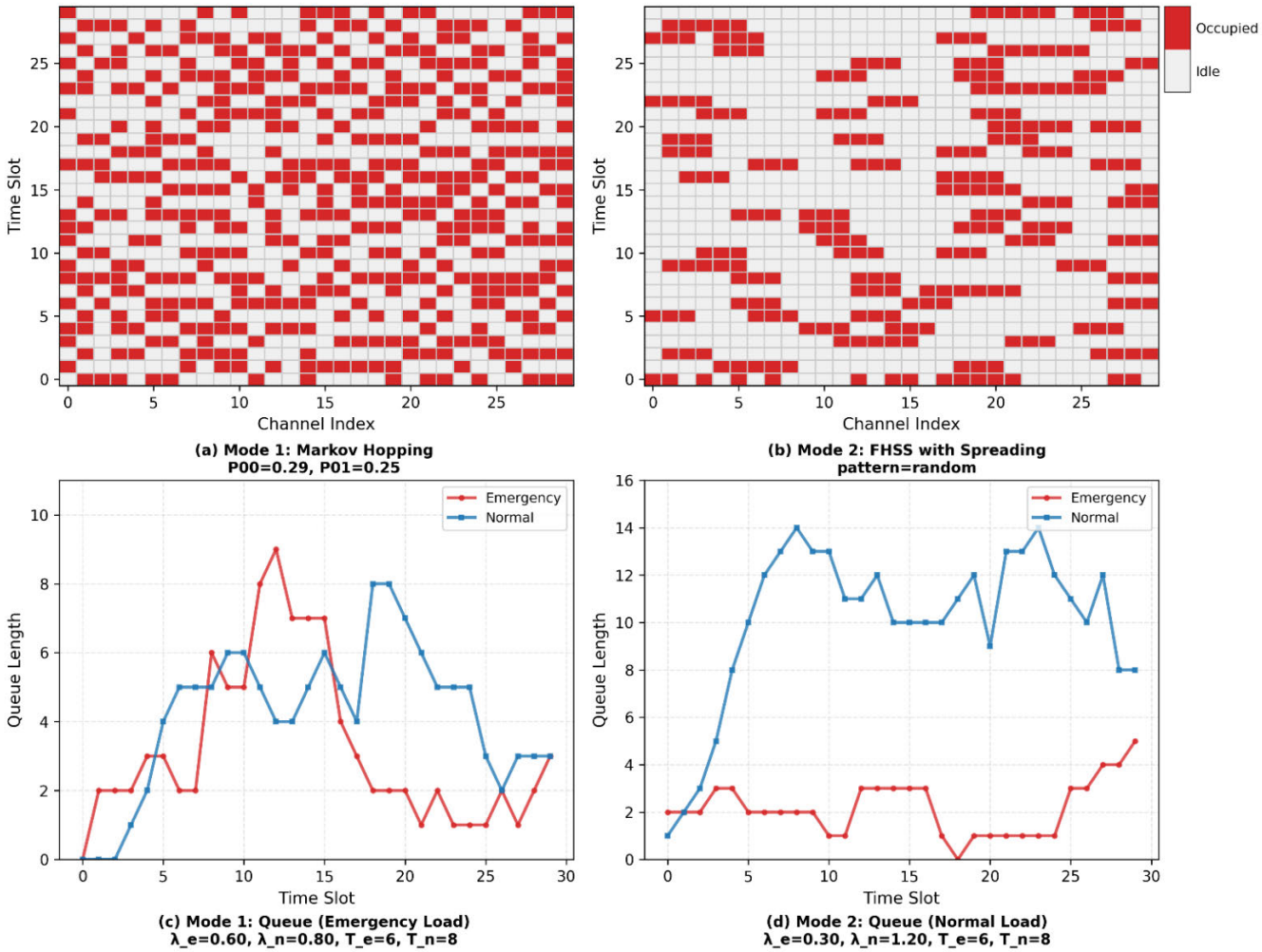

Fig. S1 Channel occupancy patterns and queue variations of two spectrum models for training phase

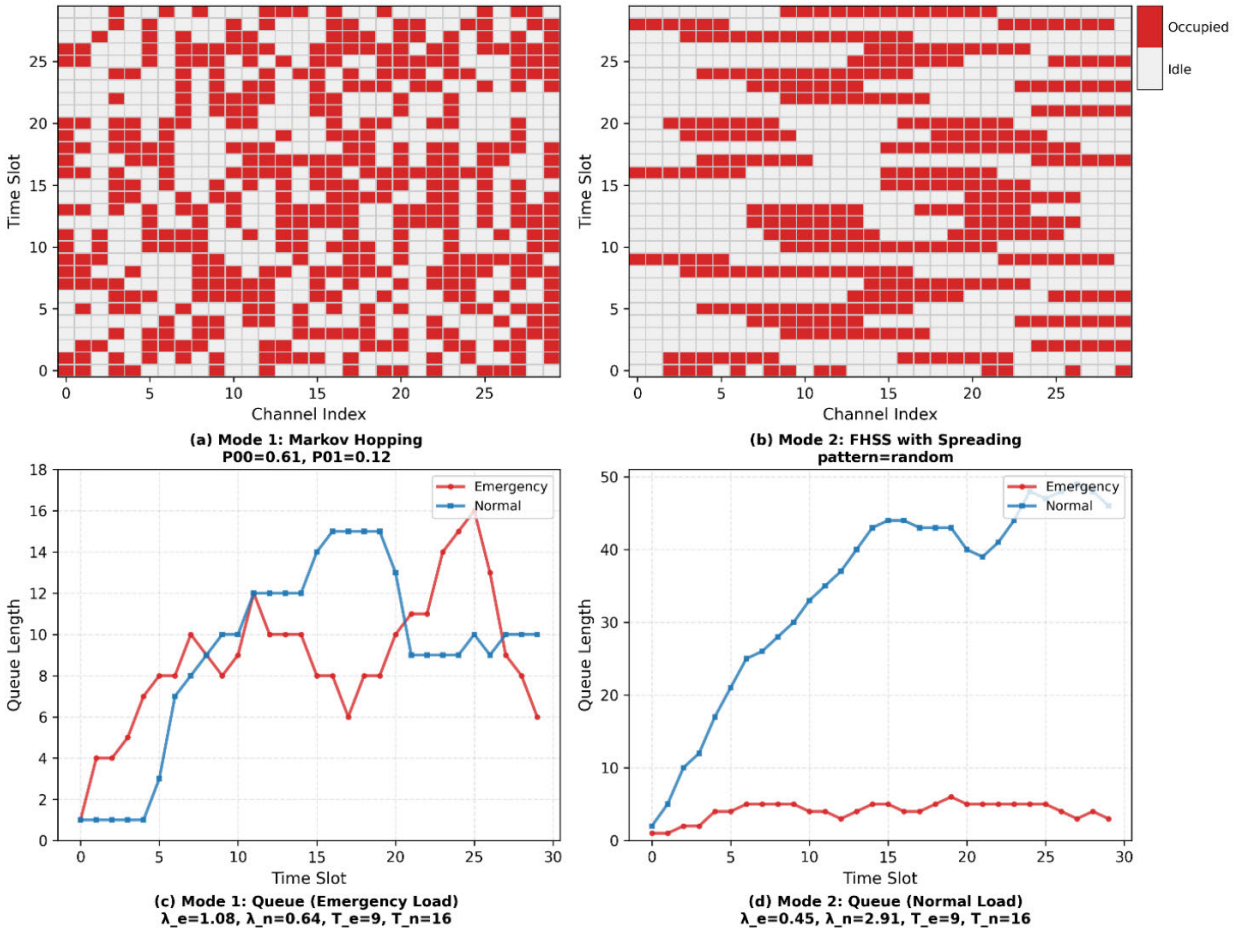

Fig. S2 Channel occupancy patterns and queue variations of two spectrum models for testing phase

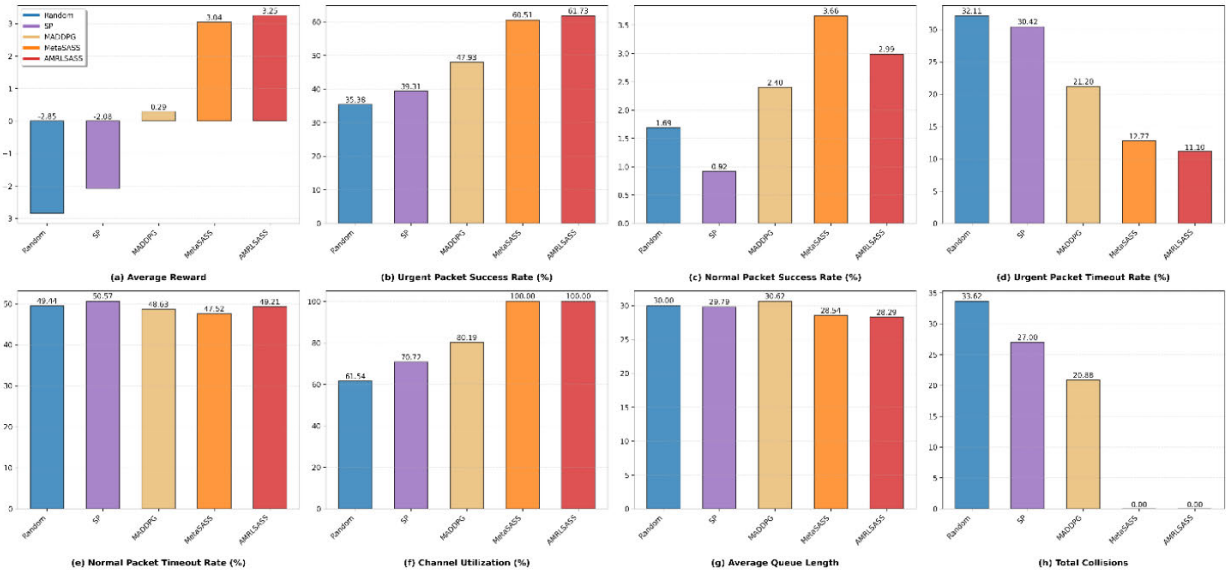

Fig. S3 Performance Comparison on Markov Models

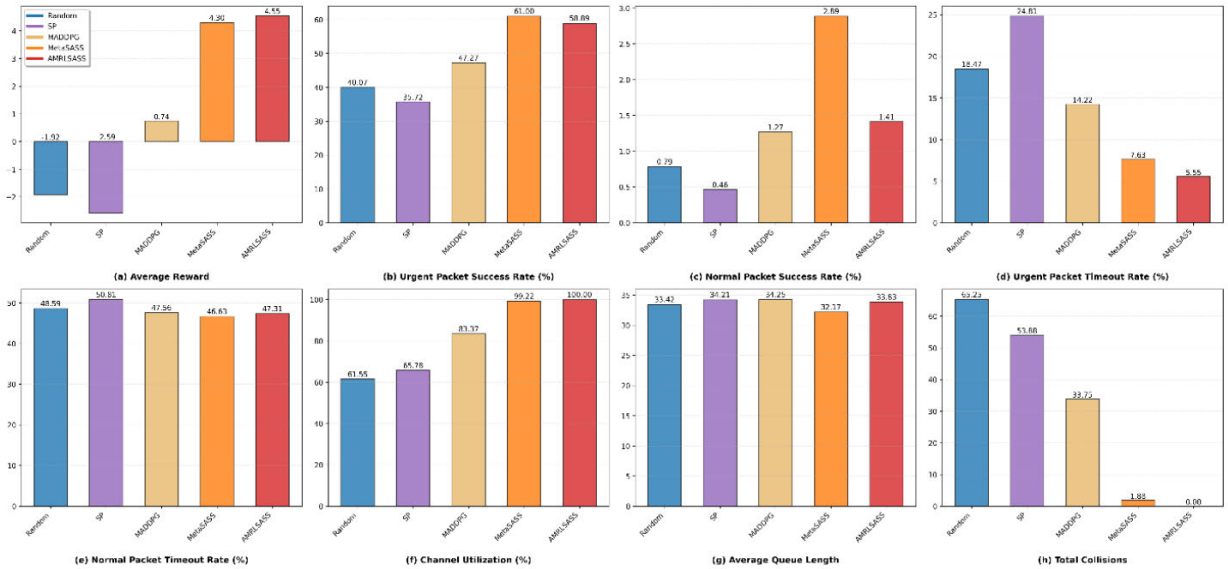

Fig. S4 Performance Comparison on FHSS Models

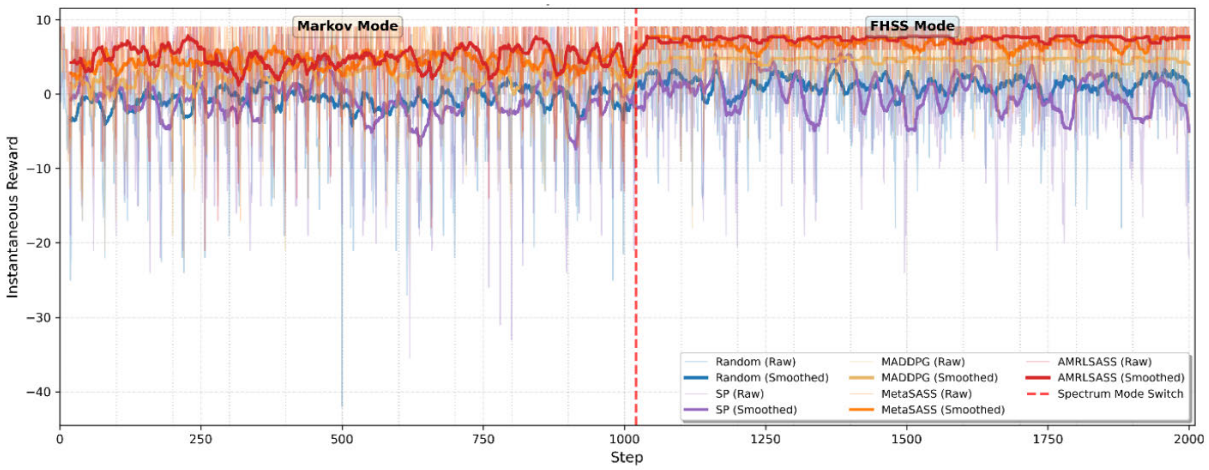

Fig. S5 Comparison of different algorithms in terms of generalization performance (step-level)

4 Scalability and Complexity Discussion

The proposed method improves decision performance by shifting most of the computational burden from per-step online inference to offline meta-training and task adaptation. In both the Markov-mode and FHSS-mode settings, the current implementation uses 20 channels, 3 decision-making agents, and sensing length 6, resulting in observation and action dimensions of 12 and 15, respectively. Each actor is implemented as a lightweight two-hidden-layer MLP with 20,111 parameters, while the complete MetaSASS and AMRLSASS policies contain 60,333 and 68,494 trainable parameters, respectively. In our current CUDA-based implementation, the joint action selection time is on the order of 1-2ms, which indicates that the online inference overhead is modest after training.

However, this improvement is achieved at the cost of higher training and adaptation complexity. In the current configuration, the fast adaptation phase requires 10 gradient update steps per task (whereas the baseline MADDPG performs no adaptation), which enables the proposed method to rapidly adjust to new environments and achieve improved cross-task adaptability. Therefore, the proposed method is most suitable for scenarios in

which higher offline computation is acceptable in exchange for low-latency online decision-making and improved cross-task adaptability—a trade-off well aligned with maritime emergency communications, where training can be performed in advance and rapid online response is paramount.

Regarding scalability, the current experiments are limited to three decision-making agents in both mode settings. Therefore, the present study should be interpreted as a proof-of-concept rather than a complete validation for dense real-world radio networks. From an algorithmic perspective, the decentralized actor-based inference employed by MetaSASS and AMRLSASS is inherently more scalable than the centralized-critic design of MADDPG as the number of users grows, because each additional user introduces only a new independent actor without expanding the input dimension of existing decision modules. Nevertheless, dedicated large-scale experimental validation remains necessary.