ASPLOS

MulBERRY: Enabling Bit-Error Robustness for Energy-Efficient Multi-Agent Autonomous Systems

Zishen Wan[1], Nandhini Chandramoorthy[2], Karthik Swaminathan[2], Pin-Yu Chen[2], Kshitij Bhardwaj[3], Vijay Janapa Reddi[4], Arijit Raychowdhury[1]

Georgia Tech

IBM

Lawrence Livermore National Laboratory

HARVARD UNIVERSITY

ASPLOS 2024

Executive Summary

Through this talk, we will:

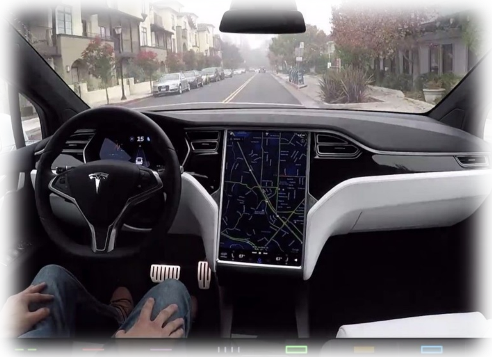
- **Understand** the challenges of building efficient and resilient swarm autonomous systems
- **Optimize** the efficiency-performance-resilience of swarm intelligence via algorithm-hardware-system co-design approach

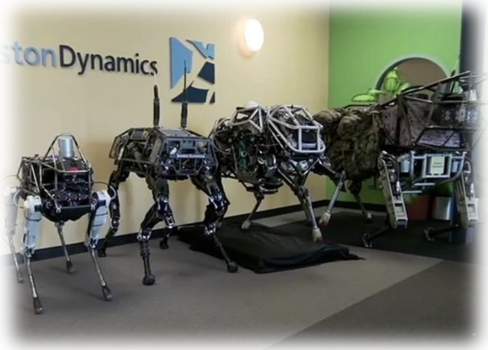
Motivation

Reliability

Goal: Improve operational resiliency
(Temporal/Spatial Redundancy)

Swarm Autonomous Machines





Performance

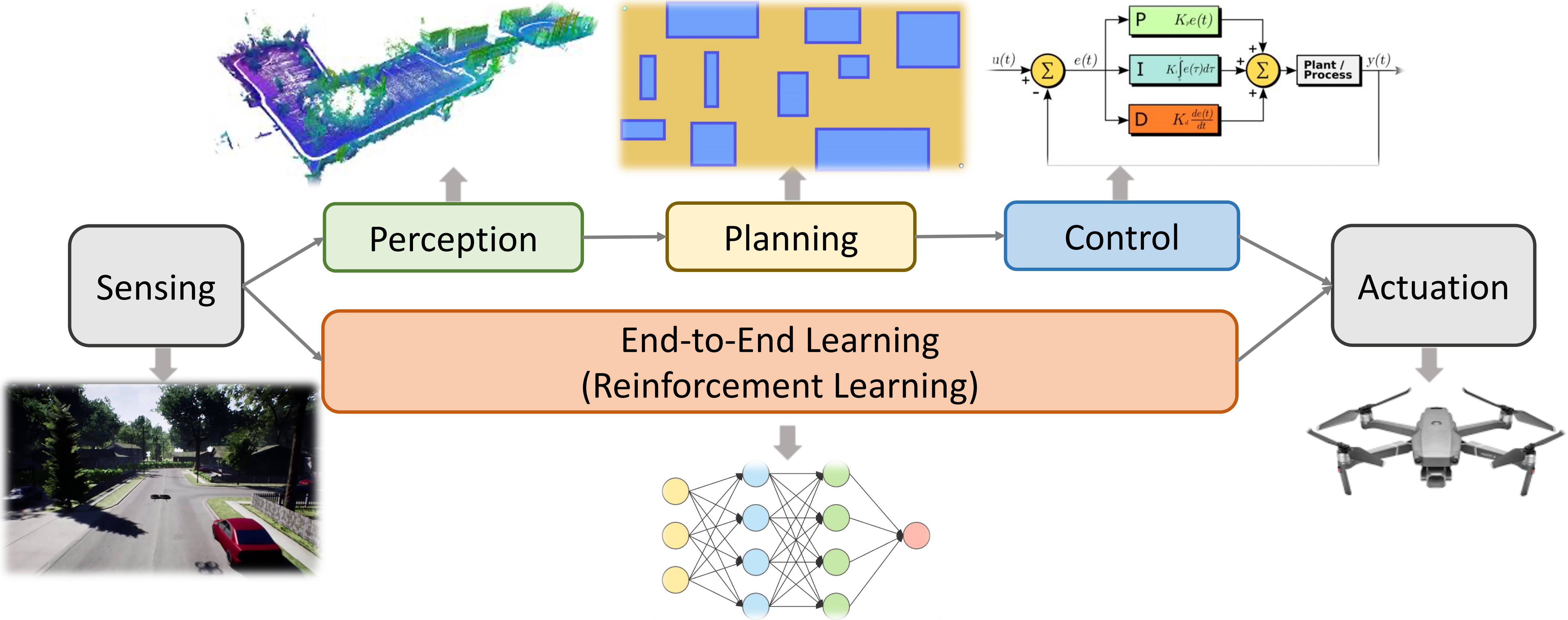
Performance-Efficiency-Reliability
Co-Optimization

Efficiency

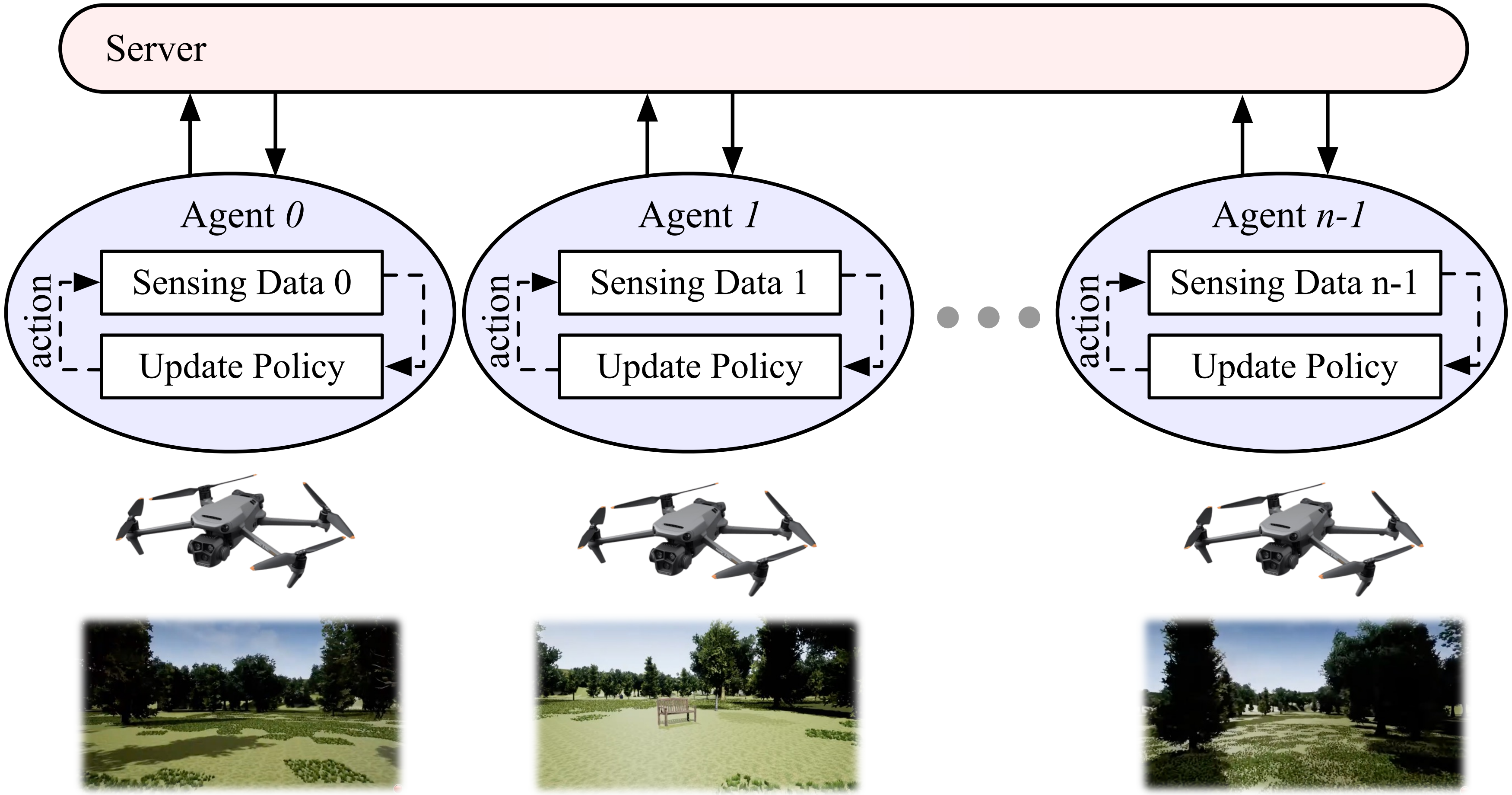
Goal: Improve mission quality
(Autonomy Algorithms)

Goal: Improve compute efficiency
(Software Optimization,
Hardware Architecture)

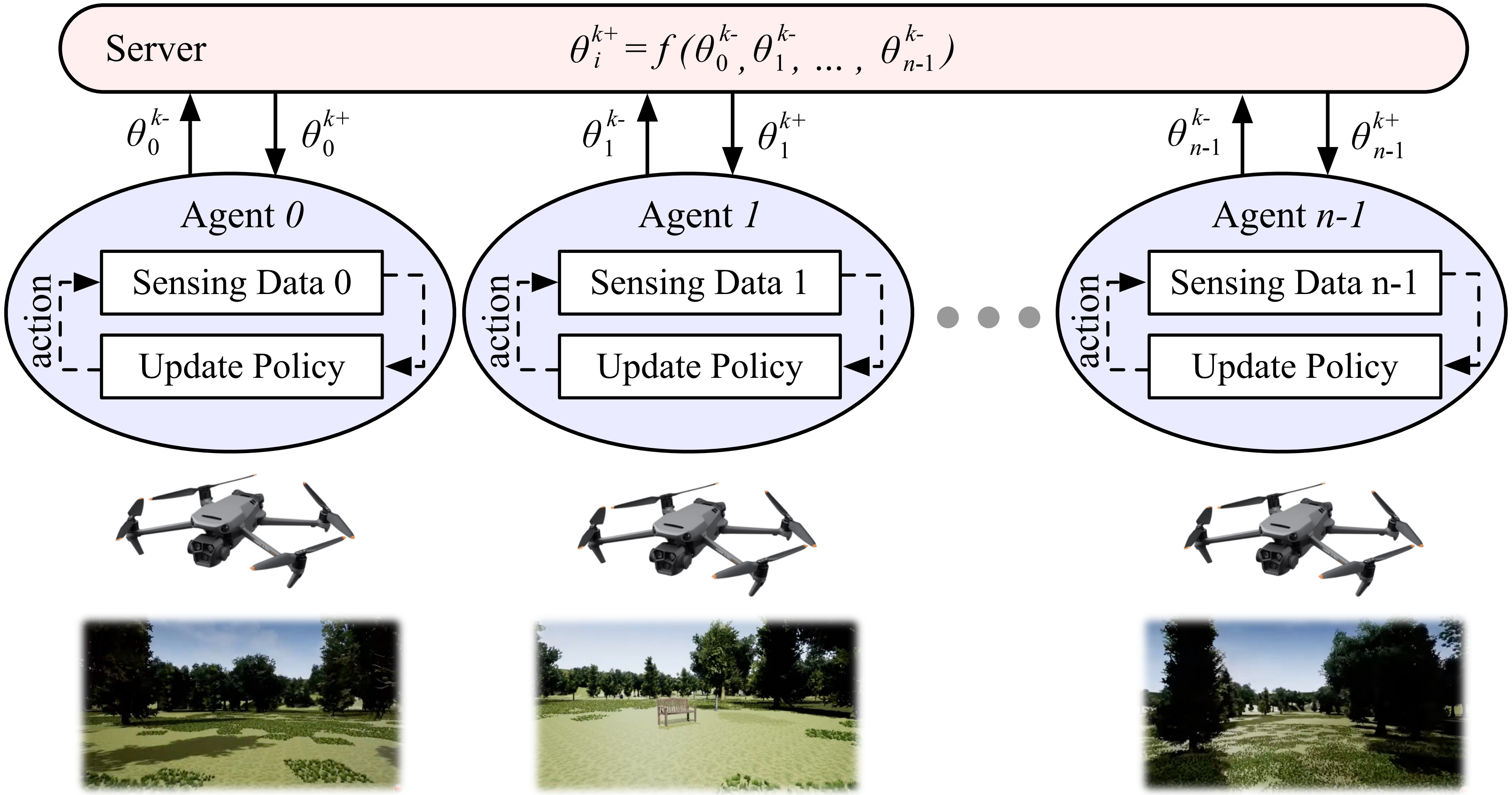
What is Autonomous Machine System?



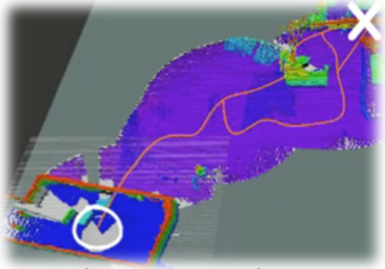
What is Swarm Autonomous Machine System?

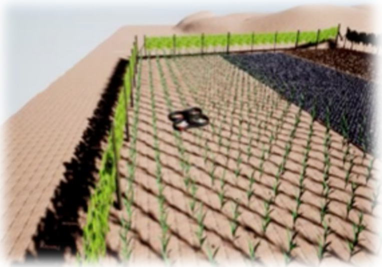


What is Swarm Autonomous Machine System?

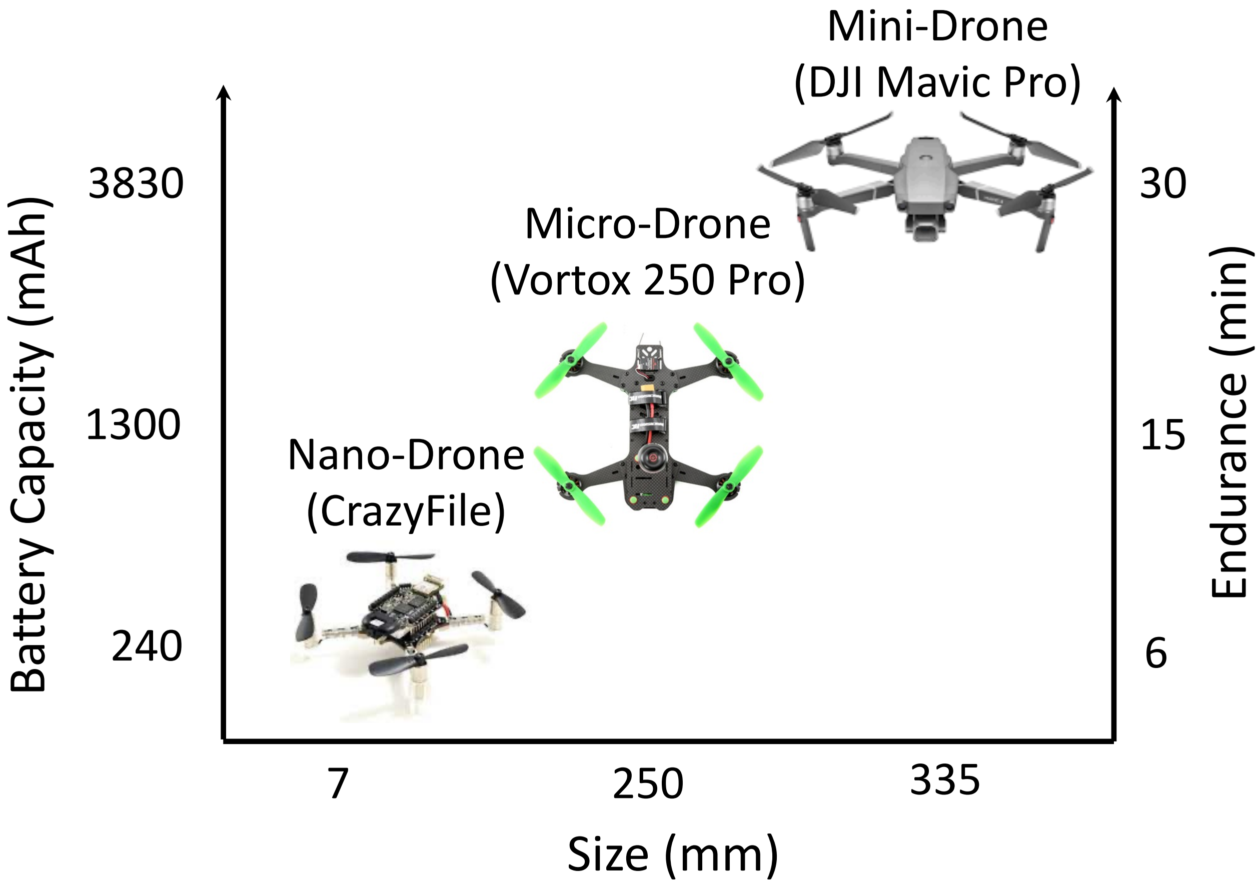


Challenge 1: Strict Resource Budgets


Package Delivery


Scanning

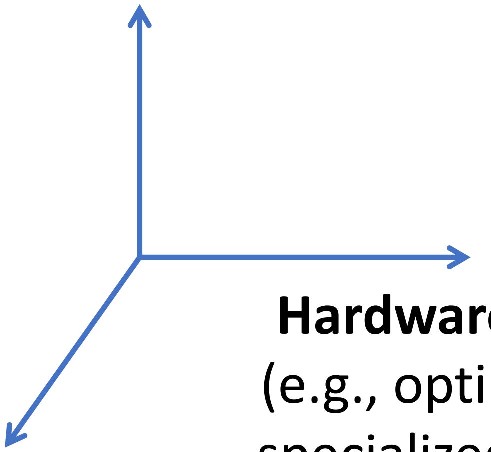

Search and Rescue



Drones are size, weight, and power (SWaP) constrained

Energy-Efficient Autonomous System

Software Optimization
(e.g., quantization, sparsity)

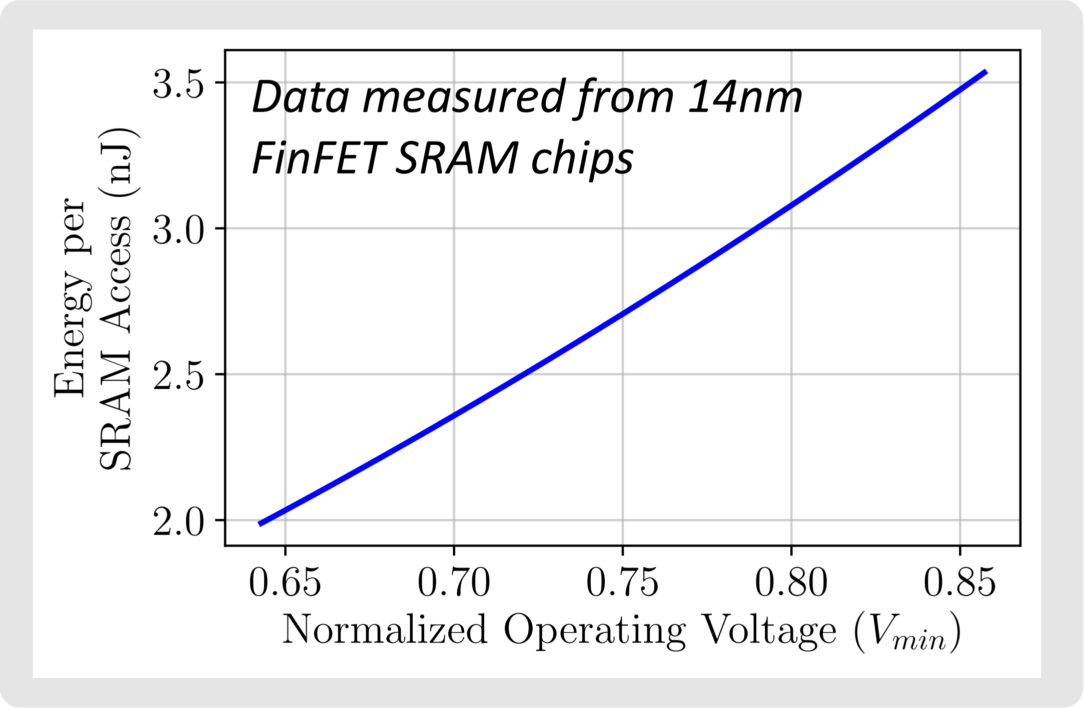


Hardware Optimization
(e.g., optimized dataflow, specialized compute unit)

Lower processor operating voltage

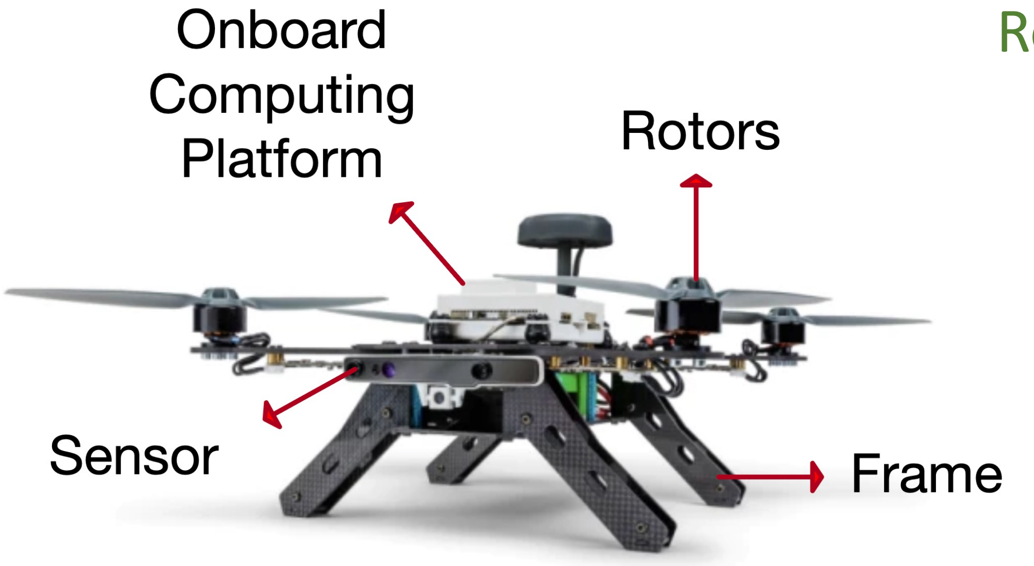
Energy ∝ Voltage ^{2}

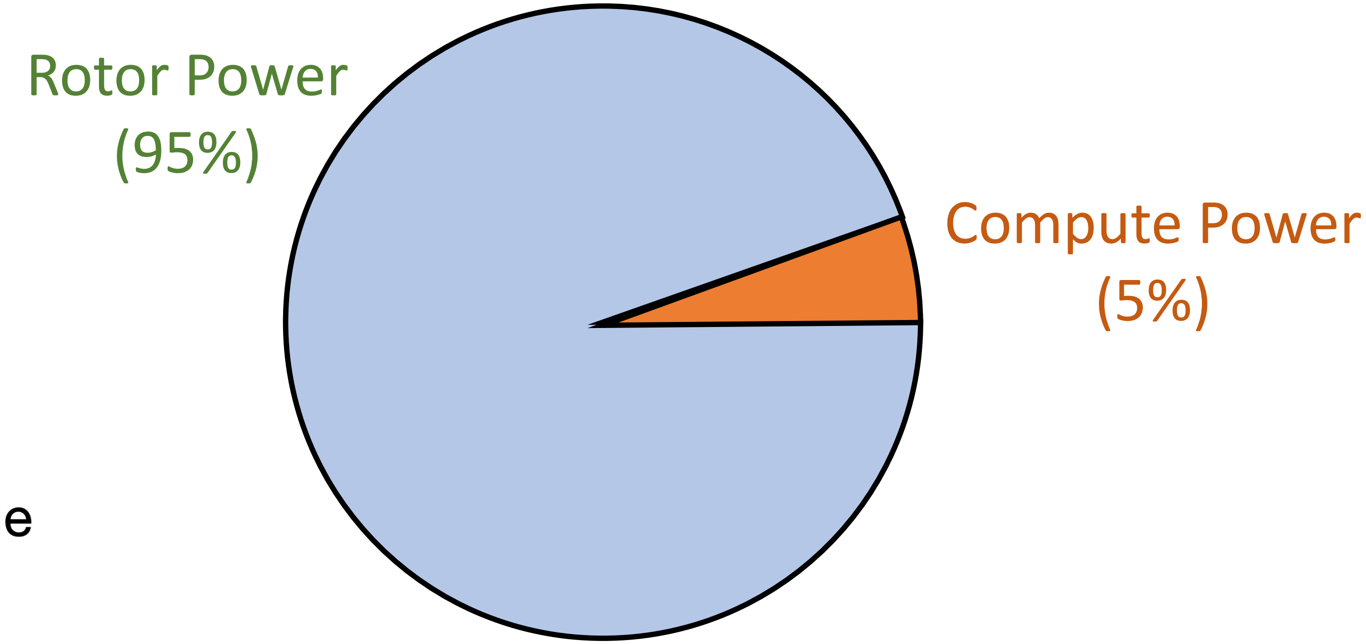
SRAM Access Energy vs. Operating Voltage



Lower operating voltage quadratically reduces energy

Challenge 2: Compute-Physics Correlation



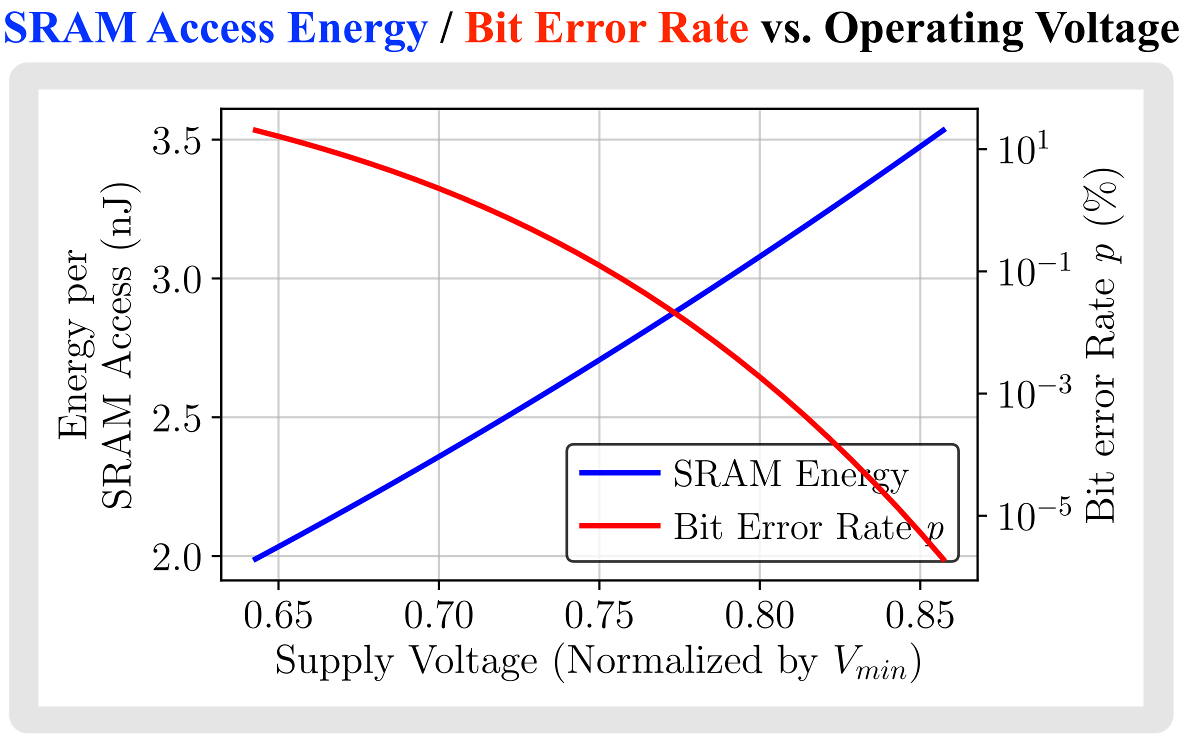


Power breakdown measured from 3DR Solo drone

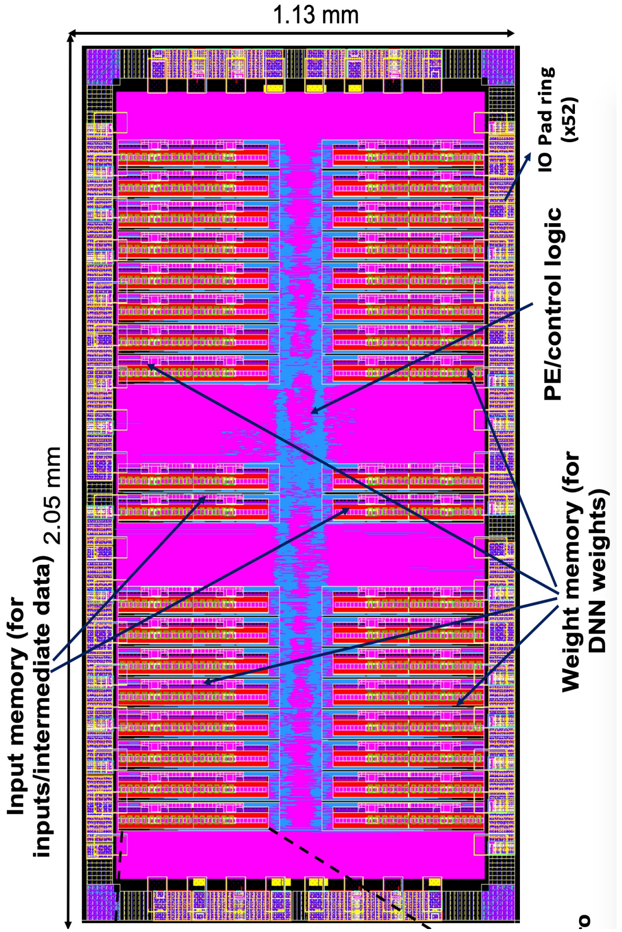
Compute power is only a small fraction of total drone power
-> Will optimize compute bring system energy-savings?

Challenge 3: Low Voltage Induces Faults

Technology	14nm
Chip Dimension	2.05 mm x 1.13 mm
Memory Capacity	128 KB weight, 16 KB input
Frequency/ Voltage	330 MHz for V_{dd}=0.8V

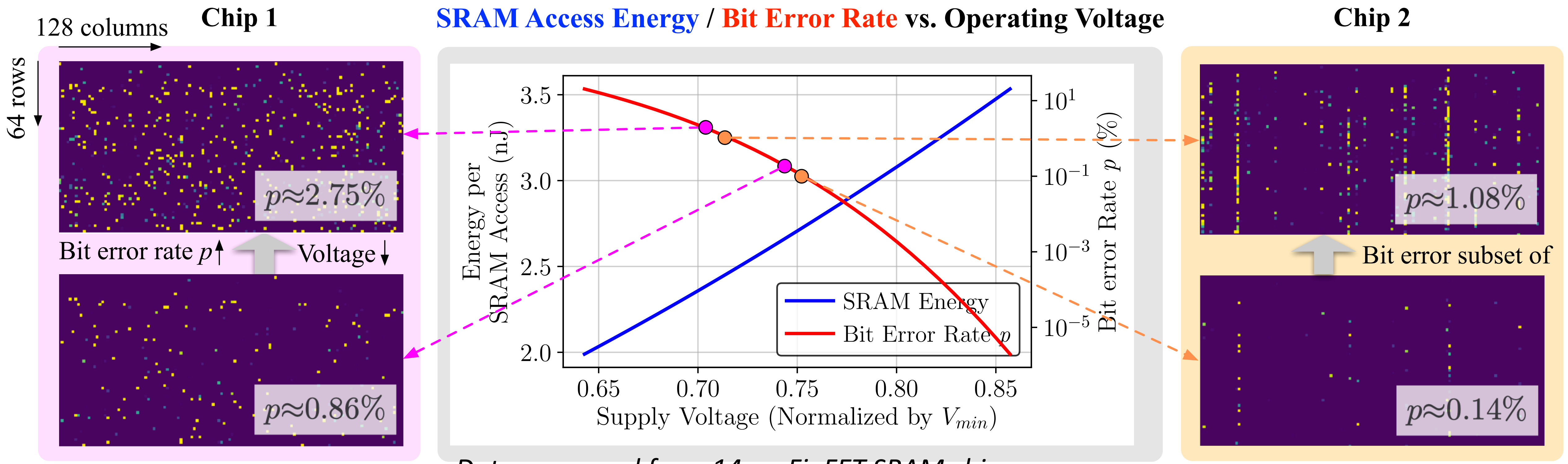


Data measured from 14nm FinFET SRAM chips



[HPCA19] Resilient Low Voltage Accelerators for High Energy Efficiency
[MLSys21] Bit Error Robustness for Energy-Efficient DNN Accelerators

Challenge 3: Low Voltage Induces Faults



Data measured from 14nm FinFET SRAM chips

Operating below rated voltage range results in memory bit errors, negatively impacting safety

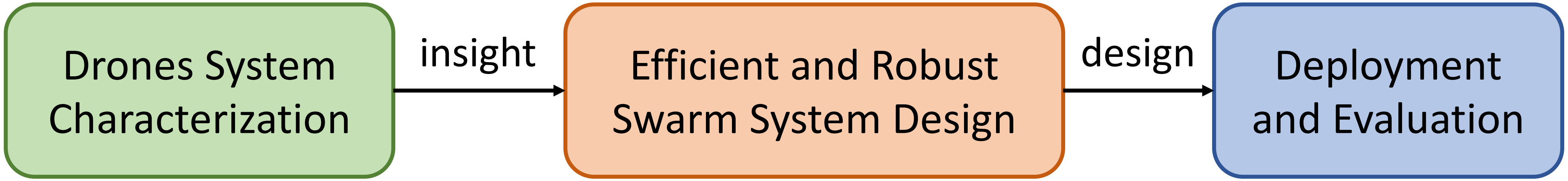
MulBERRY

~~How can~~ we achieve aggressive energy-savings under low-voltage operation, yet remain computationally-resilient for swarm autonomous systems?

(***performance-efficiency-resilience*** co-optimization)

MulBERRY Framework

(MulBERRY: Enabling Bit-Error Robustness for Energy-Efficient Multi-Agent Autonomous Systems)

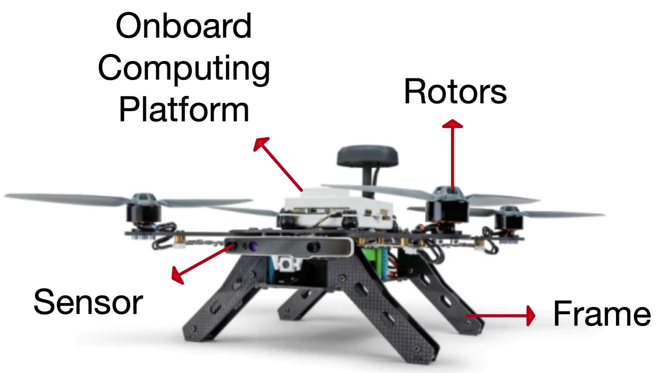


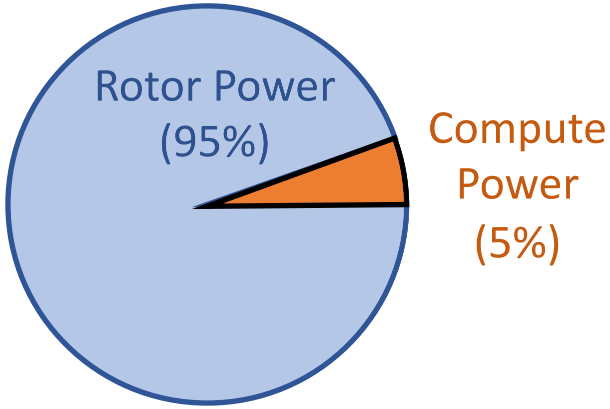
MulBERRY Framework

(MulBERRY: Enabling Bit-Error Robustness for Energy-Efficient Multi-Agent Autonomous Systems)

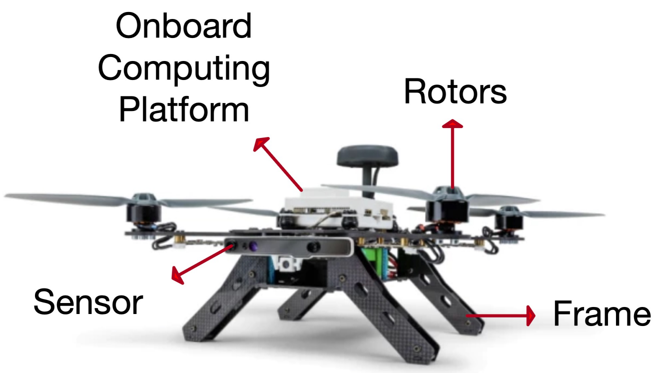
Drones System Characterization $\xrightarrow{\text{insight}}$ Efficient and Robust Swarm System Design $\xrightarrow{\text{design}}$ Deployment and Evaluation

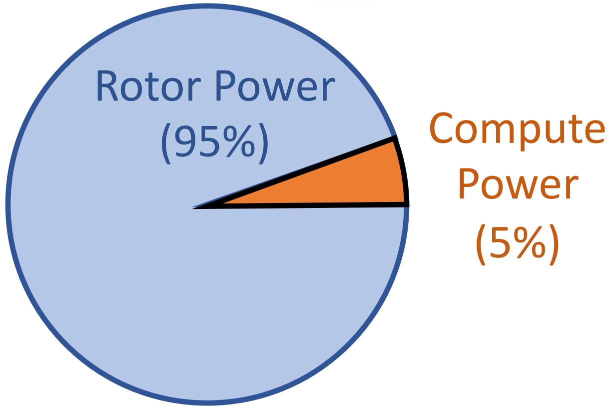
Small Compute Power, Large System Impact!

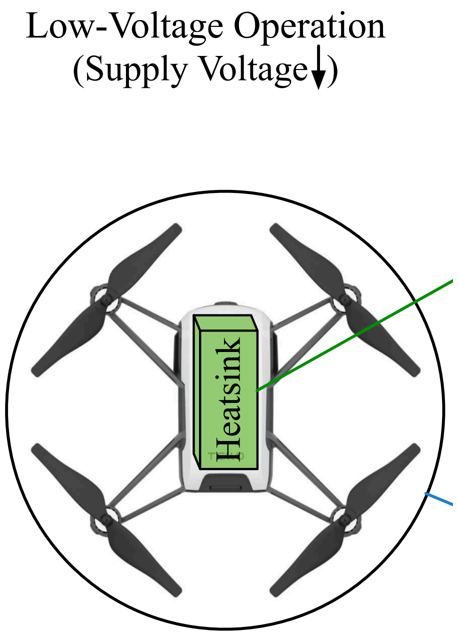




Small Compute Power, Large System Impact!

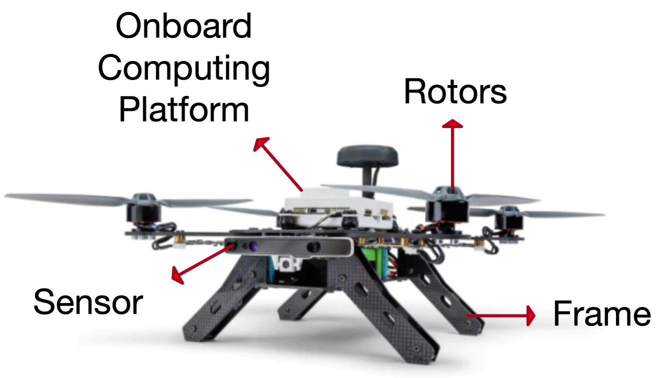


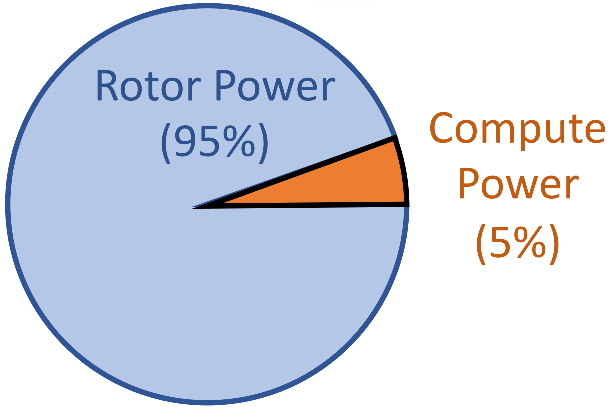


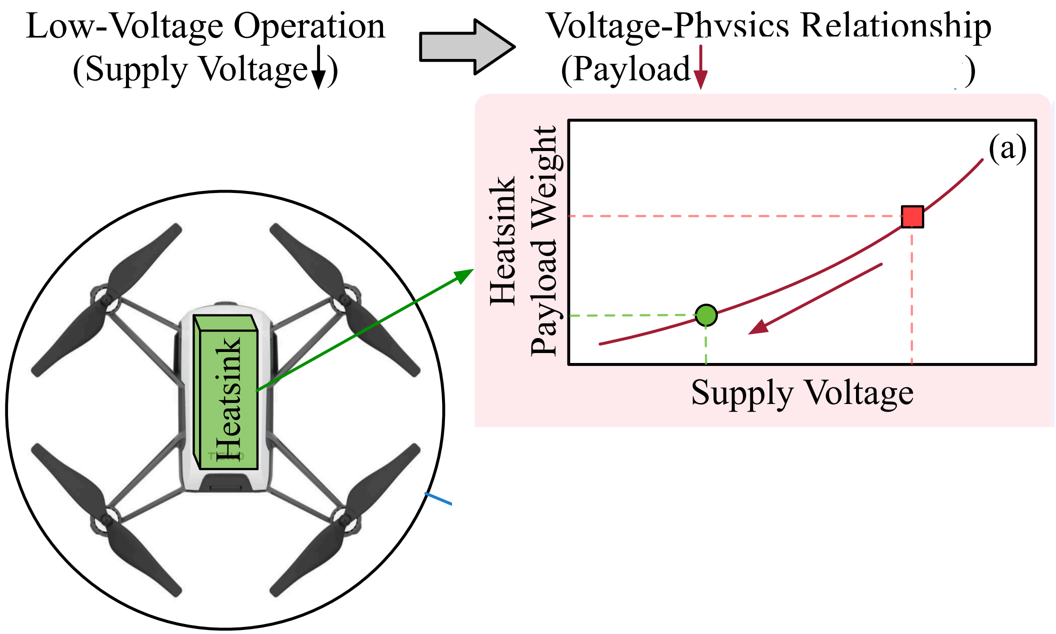


Low-voltage operation

Small Compute Power, Large System Impact!







HotSpot analysis[1] + heatsink modeling[2]

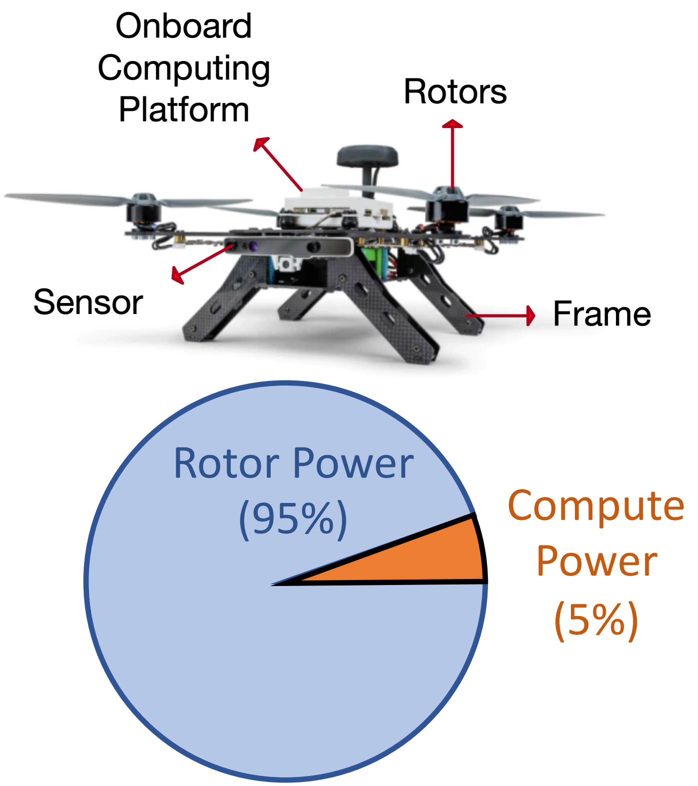
[1] Hotspot 6.0: Validation, acceleration and extension

[2] Celsia Heatsink Size Simulator

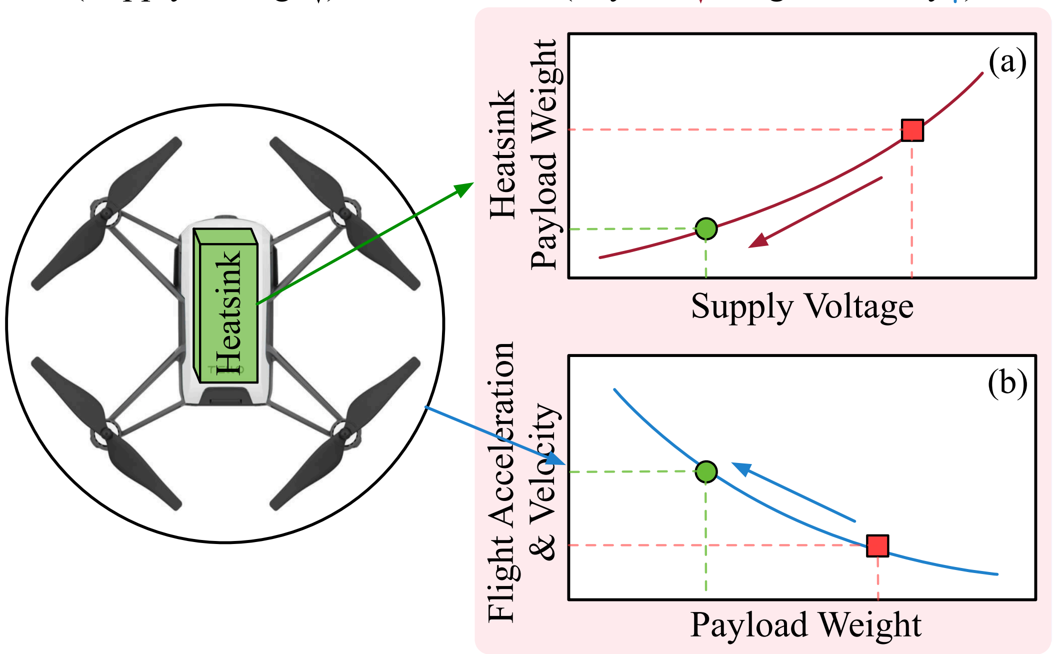
Low-voltage operation ➔ **Payload weight↓**

(Peak temperature↓, heatsink size and weight↓)

Small Compute Power, Large System Impact!



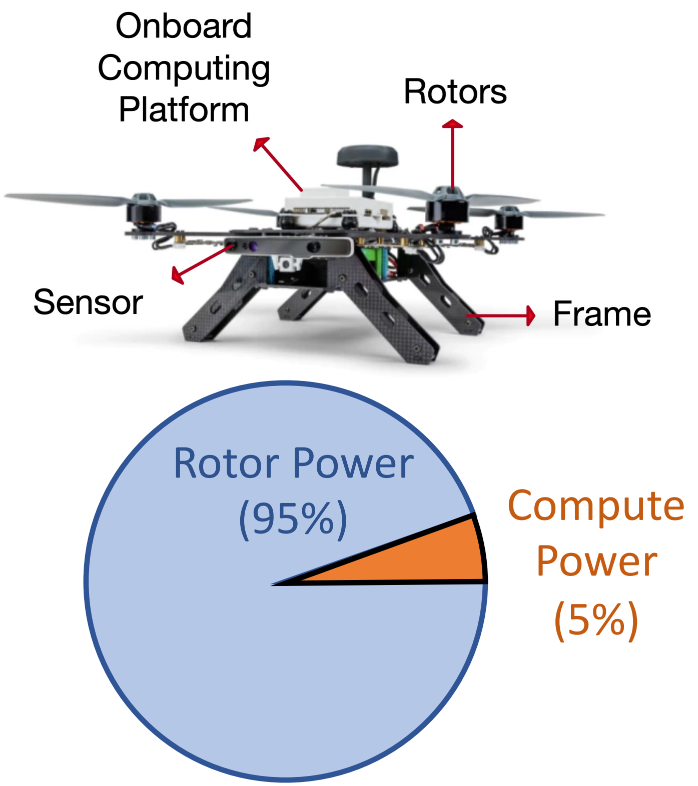
Low-Voltage Operation (Supply Voltage↓) → Voltage-Physics Relationship (Payload↓, Flight Velocity↑)

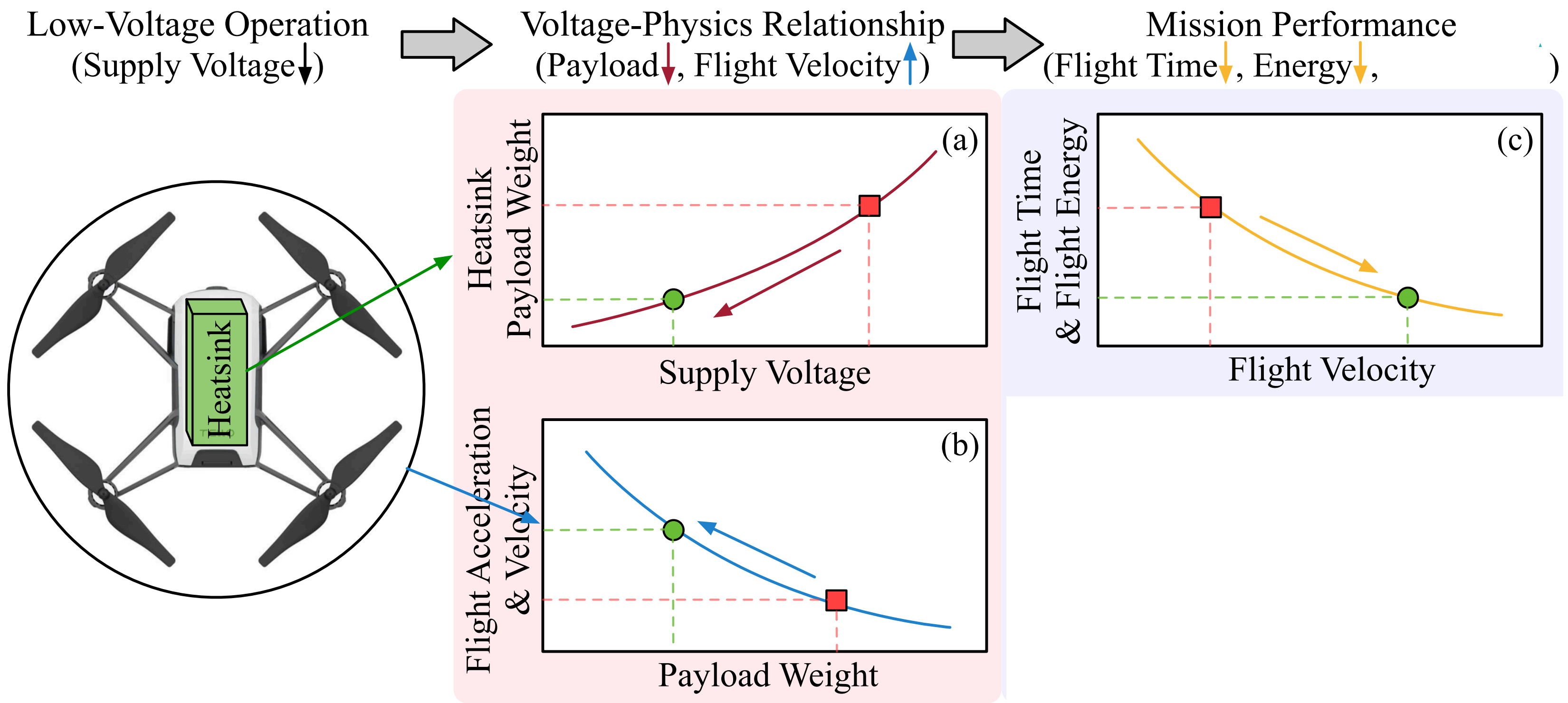


Low-voltage operation → Payload weight ↓

Safe flight velocity↑

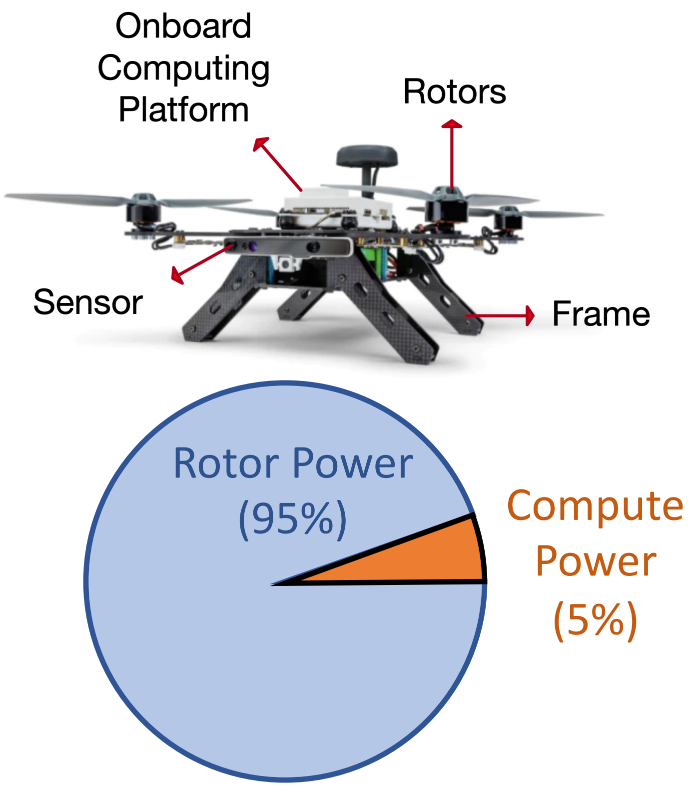
Small Compute Power, Large System Impact!

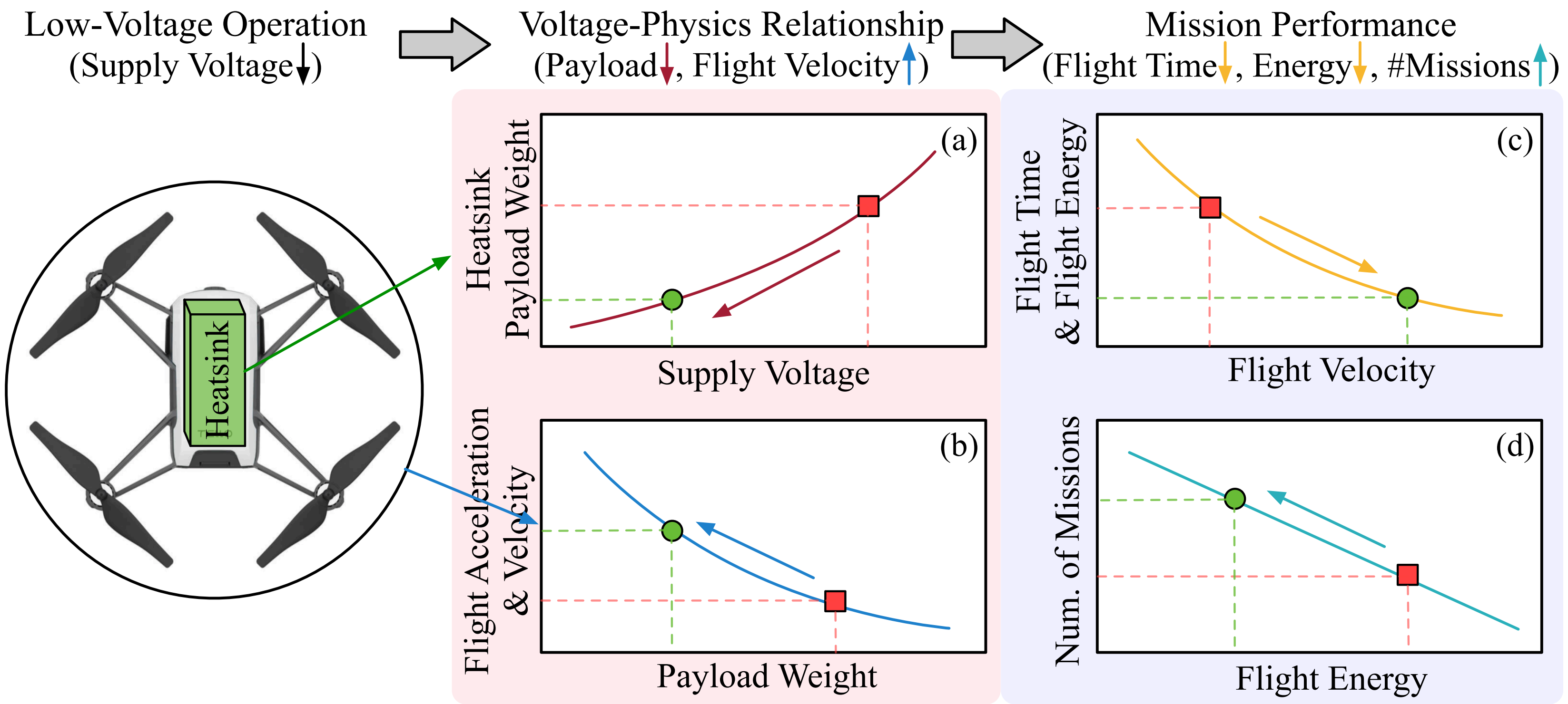




Low-voltage operation → Payload weight ↓
Safe flight velocity ↑
→ **Flight time & energy↓**

Small Compute Power, Large System Impact!

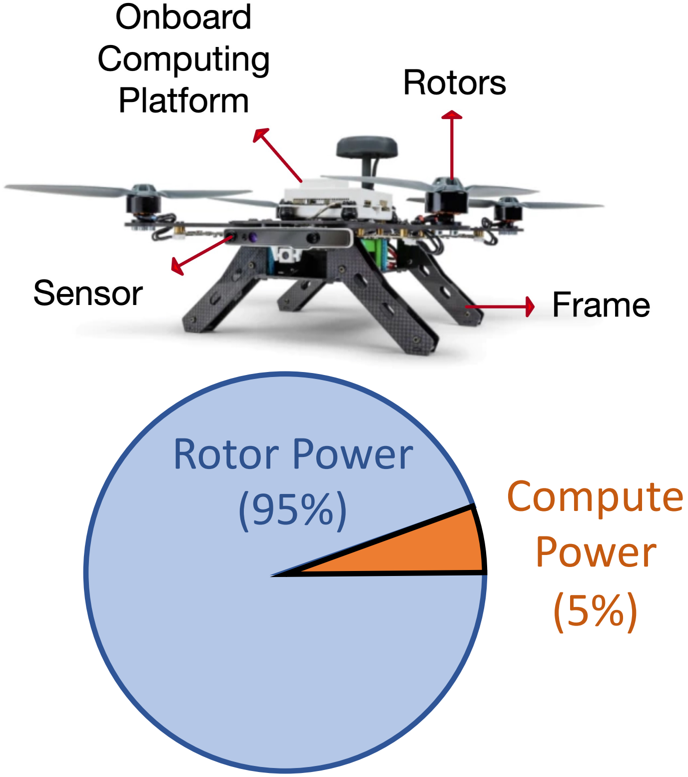


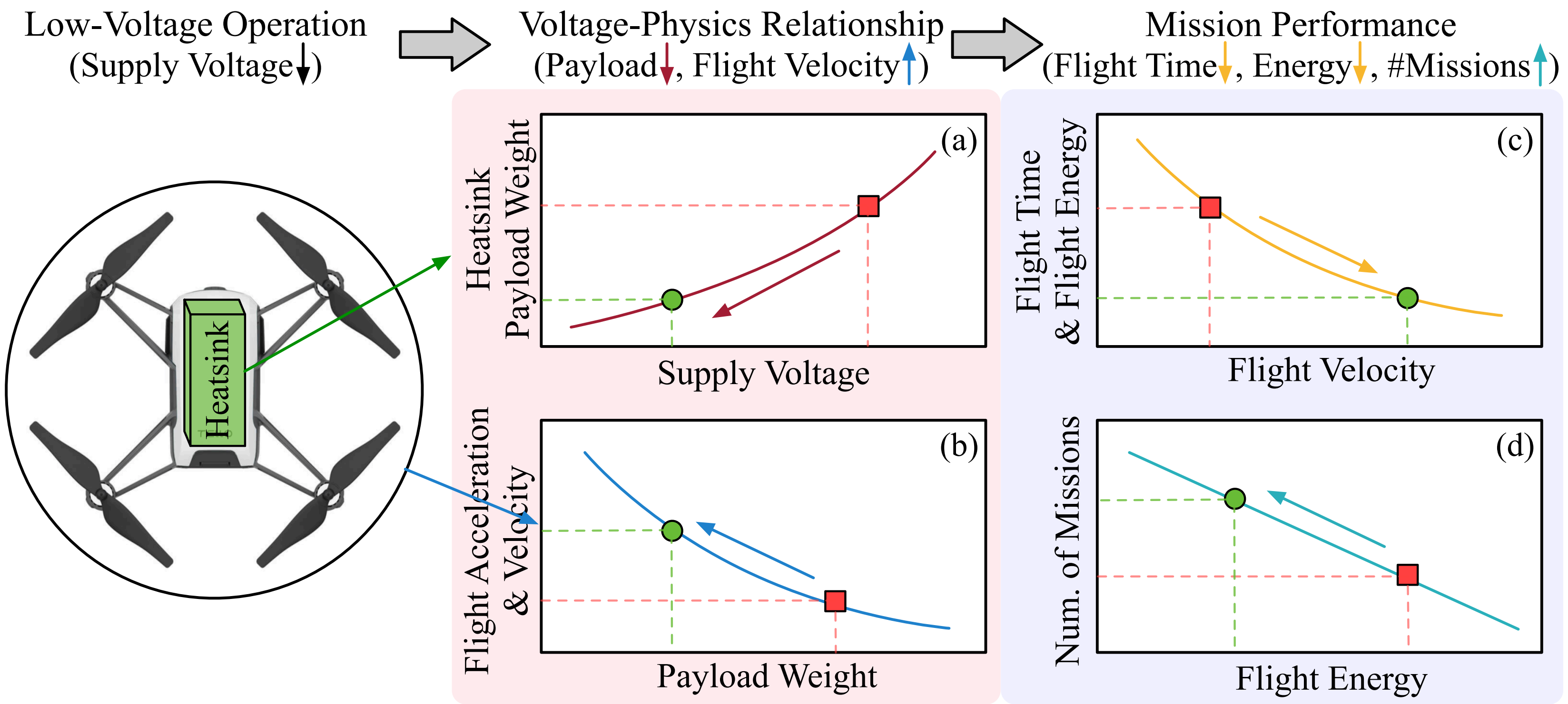


Low-voltage operation → Payload weight ↓ → Flight time & energy ↓

Safe flight velocity ↑ → **Number of missions ↑**

Small Compute Power, Large System Impact!





Compute power has huge impacts on end-to-end autonomous system mission energy

MulBERRY Framework

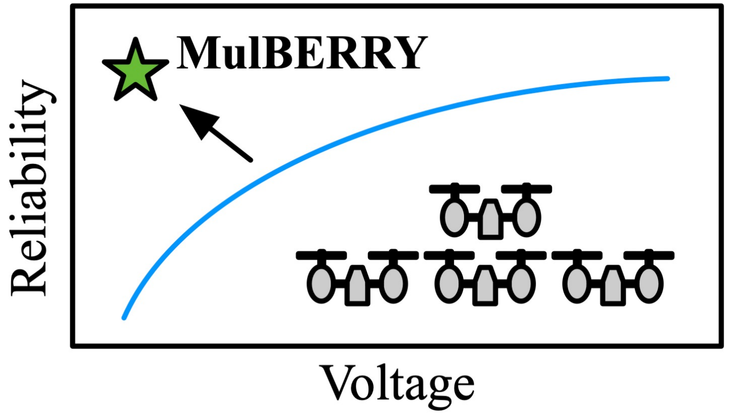
(MulBERRY: Enabling Bit-Error Robustness for Energy-Efficient Multi-Agent Autonomous Systems)

MulBERRY System Design Principle

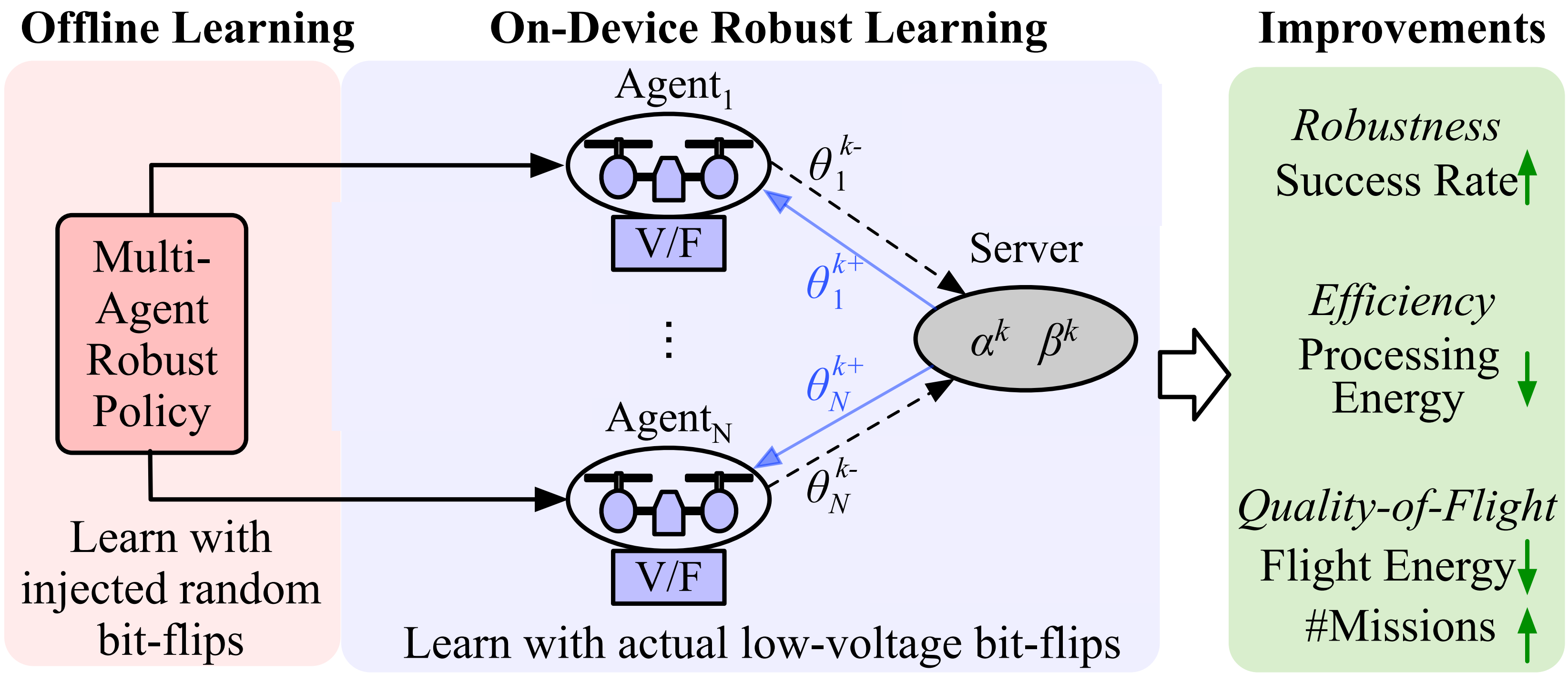
- **Design Principle**: Cross-layer swarm robust learning framework, integrates *algorithm-level* error-aware learning with *system-level* collaborative server-agent optimization and *hardware-level* thermal-voltage adaptive adjustment.

MulBERRY Objective

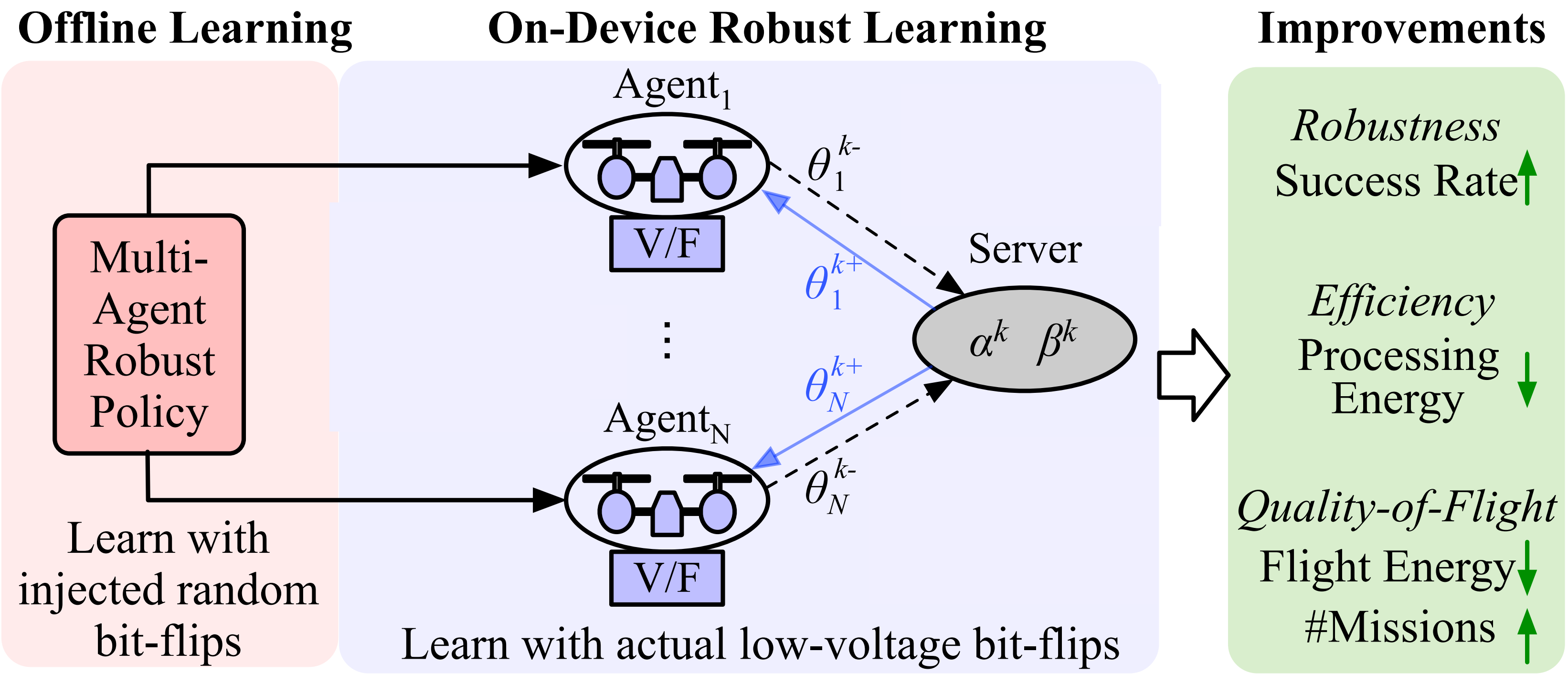
- **<u>Design Principle</u>**: Cross-layer swarm robust learning framework, integrates *algorithm-level* error-aware learning with *system-level* collaborative server-agent optimization and *hardware-level* thermal-voltage adaptive adjustment.
- **<u>Achieve</u>**: Aggressive *energy-savings* under *low-voltage operation*, yet *computationally-resilient* for swarm autonomous systems.



MulBERRY Key Techniques



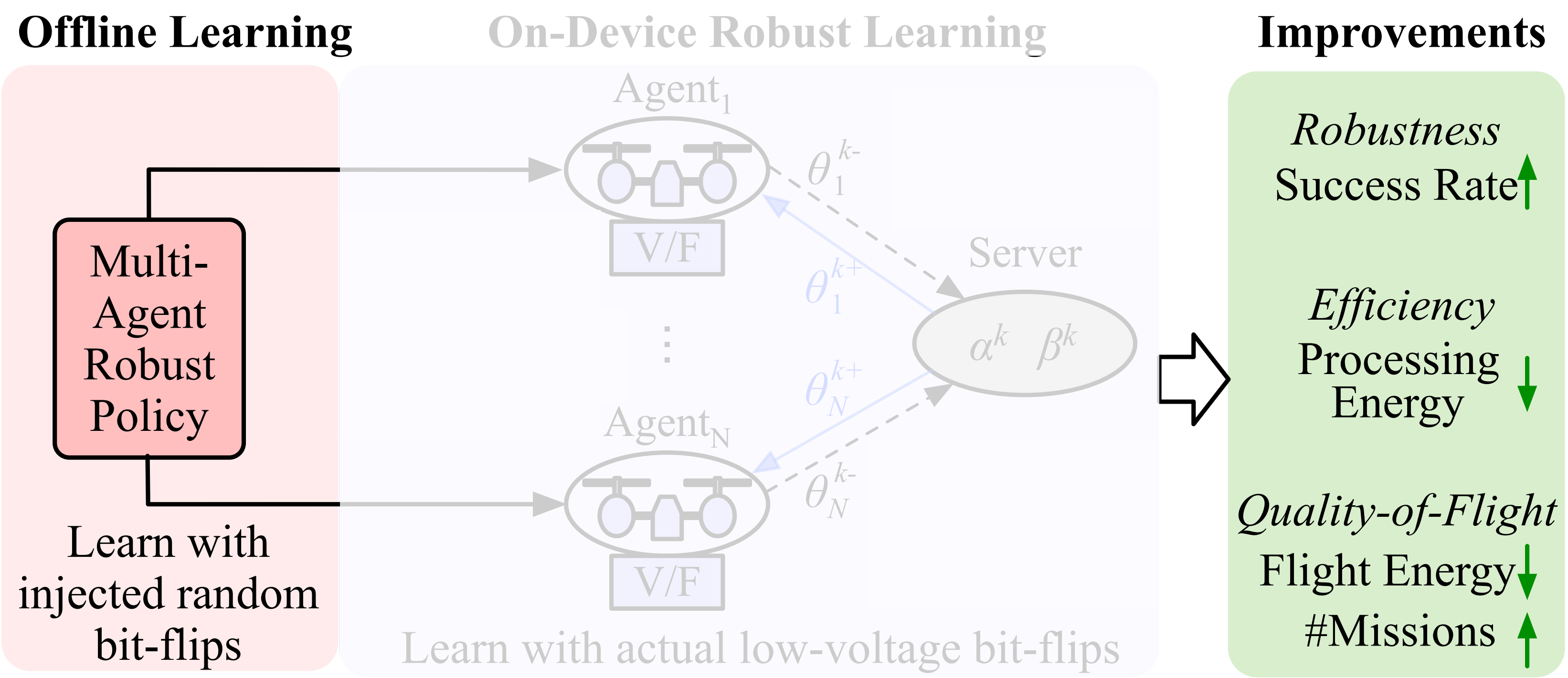
MulBERRY Key Techniques



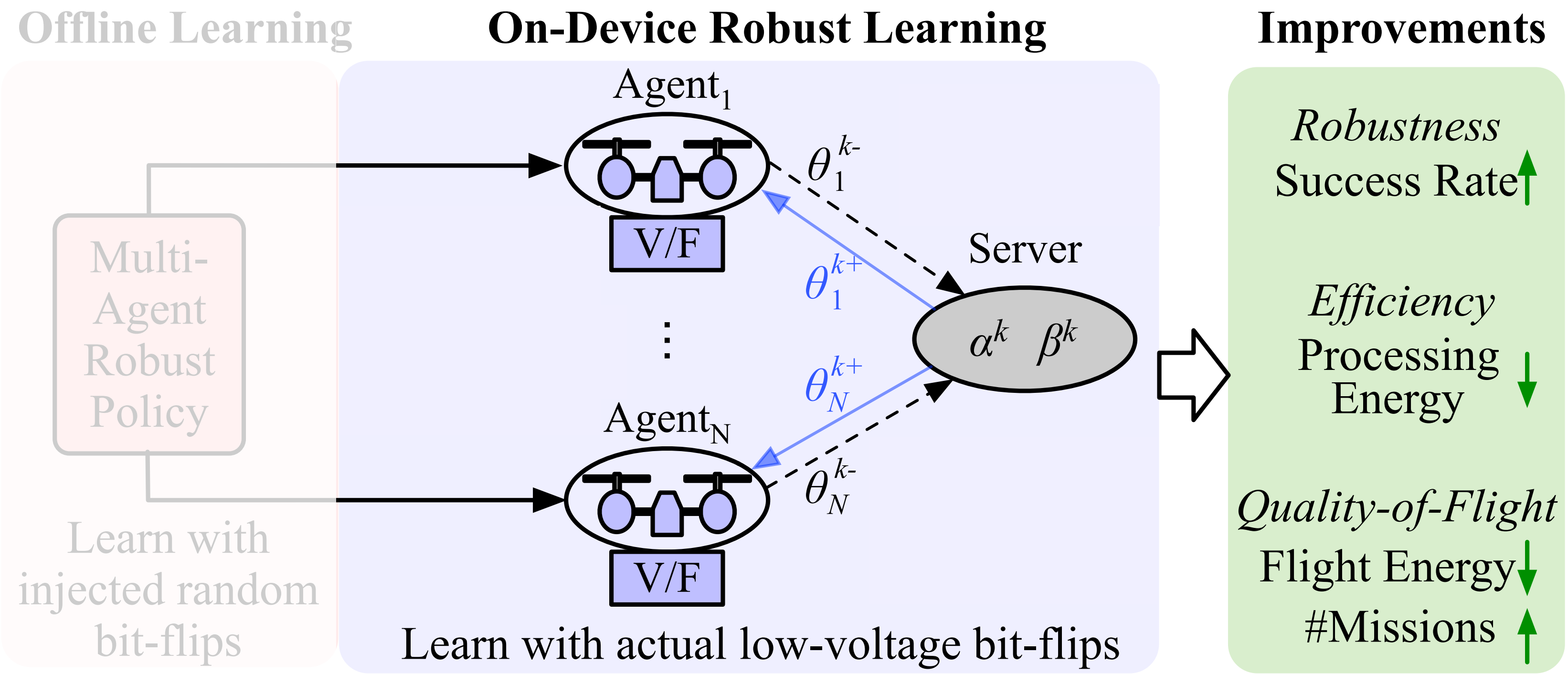
Two-Stage Swarm Robust Learning

MulBERRY Key Techniques

Two-Stage Swarm Robust Learning



MulBERRY Key Techniques



Two-Stage Swarm Robust Learning

MulBERRY Framework

Algorithm 2 MulBERRY

Initialization: number of agent n, communication interval CI, smoothing average threshold δ^k. For each agent, initialize action-value function Q with policy θ_i and target action-value function $\hat{Q}$ with policy $\theta^p = \theta$
for *time step* $k = 1$ *to* T **do**
  // Agents conduct bit-flip robust learning at each step
  for *each agent* i **in parallel do**
    Update $\theta_i^k \leftarrow \texttt{BitFlipLearning}\left(i, \theta_i^{(k-1)}\right)$
  end for
  // Agents communicate with server at every CI steps
  if $k \bmod CI = 0$ **then**
    Each agent i sends policy θ_i^{k-} to server
    Server calculates smoothing average parameters:
    $\alpha^k = \frac{1}{n}\max(1, \frac{(1-n)k}{\delta^k} + n)$, $\beta^k = \frac{1-\alpha^k}{n-1}$
    for *each agent* i **do**
      Server sends its updated policy θ_i^{k+} back to agent i:
      $\theta_i^{k+} = \alpha^k \theta_i^{k-} + \beta^k \sum_{i \neq j} \theta_j^{k-}$
Function: $\texttt{BitFlipLearning}$ $(i, \theta^{(k)})$
 Given state s_k, take action a_k based on Q (ϵ-greedy)
 Obtain reward r_k and reach new state s_{k+1}
 Store transition (s_k, a_k, r_k, s_{k+1}) in D
 // Experience replay
 Sample a mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{b=1}^{B}$ from D
 // Clean training pass
 Set $y_j = r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta^{p(k)})$
 $\Delta^{(k)} = \nabla_\theta \sum_{b=1}^{B} (Q(s_j, a_j; \theta^{(k)}) - y_j)^2$
 // Perturbed training pass, inject bit errors at rate p
 $\tilde{\theta}^{(k)} = BErr_p(\theta^{(k)})$
 Set $\tilde{y}_j = (r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \tilde{\theta}^{p(k)}))$
 $\tilde{\Delta}^{(k)} = \nabla_\theta \sum_{b=1}^{B} (Q(s_j, a_j; \tilde{\theta}^{(k)}) - \tilde{y}_j)^2$
 // Average gradients and update w.r.t θ
 $\theta^{(k+1)} = \theta^{(k)} - \alpha(\Delta^{(k)} + \tilde{\Delta}^{(k)})$
 // Periodic update of target network
 Every C steps reset $\hat{Q} = Q$, i.e., set $\theta^p = \theta$
Return $\theta^{(k+1)}$

Output: Unified multi-agent bit-error robust policy θ

MulBERRY Framework

Start: Initialize swarm autonomy model

Goal: learn **robust** swarm autonomy model

Algorithm 2 MulBERRY

Initialization: number of agent n, communication interval CI, smoothing average threshold δ^k. For each agent, initialize action-value function Q with policy θ_i and target action-value function $\hat{Q}$ with policy $\theta^p = \theta$

for *time step* $k = 1$ *to* T **do**

  // Agents conduct bit-flip robust learning at each step

  for *each agent* i **in parallel do**

    Update $\theta_i^k \leftarrow$ `BitFlipLearning` $\left(i, \theta_i^{(k-1)}\right)$

  end for

  // Agents communicate with server at every CI steps

  if k *mod* $CI = 0$ **then**

    Each agent i sends policy θ_i^{k-} to server

    Server calculates smoothing average parameters:

    $\alpha^k = \frac{1}{n}\max(1, \frac{(1-n)k}{\delta^k} + n),\ \beta^k = \frac{1-\alpha^k}{n-1}$

    for *each agent* i **do**

      Server sends its updated policy θ_i^{k+} back to agent i:

      $\theta_i^{k+} = \alpha^k \theta_i^{k-} + \beta^k \sum_{i \neq j} \theta_j^{k-}$

Function: `BitFlipLearning` $(i, \theta^{(k)})$

 Given state s_k, take action a_k based on Q (ϵ-greedy)

 Obtain reward r_k and reach new state s_{k+1}

 Store transition (s_k, a_k, r_k, s_{k+1}) in D

 // Experience replay

 Sample a mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{b=1}^{B}$ from D

 // Clean training pass

 Set $y_j = r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta^{p(k)})$

 $\Delta^{(k)} = \nabla_\theta \sum_{b=1}^{B} (Q(s_j, a_j; \theta^{(k)}) - y_j)^2$

 // Perturbed training pass, inject bit errors at rate p

 $\tilde{\theta}^{(k)} = BErr_p(\theta^{(k)})$

 Set $\tilde{y}_j = (r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \tilde{\theta}^{p(k)}))$

 $\tilde{\Delta}^{(k)} = \nabla_\theta \sum_{b=1}^{B} (Q(s_j, a_j; \tilde{\theta}^{(k)}) - \tilde{y}_j)^2$

 // Average gradients and update w.r.t θ

 $\theta^{(k+1)} = \theta^{(k)} - \alpha(\Delta^{(k)} + \tilde{\Delta}^{(k)})$

 // Periodic update of target network

 Every C steps reset $\hat{Q} = Q$, i.e., set $\theta^p = \theta$

Return $\theta^{(k+1)}$

Output: Unified multi-agent bit-error robust policy θ

MulBERRY Framework

Start: Initialize swarm autonomy model

↓

Agent error-aware learning; Server-agent communication

↓

Goal: learn **robust** swarm autonomy model

Algorithm 2 MulBERRY

Initialization: number of agent n, communication interval CI, smoothing average threshold δ^k. For each agent, initialize action-value function Q with policy θ_i and target action-value function $\hat{Q}$ with policy $\theta^p = \theta$

for *time step* $k = 1$ *to* T **do**

  // Agents conduct bit-flip robust learning at each step

  for *each agent* i **in parallel do**

    Update $\theta_i^k \leftarrow$ `BitFlipLearning` $\left(i, \theta_i^{(k-1)}\right)$

  end for

  // Agents communicate with server at every CI steps

  if $k \bmod CI = 0$ **then**

    Each agent i sends policy θ_i^{k-} to server

    Server calculates smoothing average parameters:

    $\alpha^k = \frac{1}{n}\max(1, \frac{(1-n)k}{\delta^k} + n)$, $\beta^k = \frac{1-\alpha^k}{n-1}$

    for *each agent* i **do**

      Server sends its updated policy θ_i^{k+} back to agent i:

      $\theta_i^{k+} = \alpha^k \theta_i^{k-} + \beta^k \sum_{i \neq j} \theta_j^{k-}$

Function: `BitFlipLearning` $(i, \theta^{(k)})$

Given state s_k, take action a_k based on Q (ϵ-greedy)

Obtain reward r_k and reach new state s_{k+1}

Store transition (s_k, a_k, r_k, s_{k+1}) in D

// Experience replay

Sample a mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{b=1}^{B}$ from D

// Clean training pass

Set $y_j = r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta^{p(k)})$

$\Delta^{(k)} = \nabla_\theta \sum_{b=1}^{B} (Q(s_j, a_j; \theta^{(k)}) - y_j)^2$

// Perturbed training pass, inject bit errors at rate p

$\tilde{\theta}^{(k)} = BErr_p(\theta^{(k)})$

Set $\tilde{y}_j = (r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \tilde{\theta}^{p(k)}))$

$\tilde{\Delta}^{(k)} = \nabla_\theta \sum_{b=1}^{B} (Q(s_j, a_j; \tilde{\theta}^{(k)}) - \tilde{y}_j)^2$

// Average gradients and update w.r.t θ

$\theta^{(k+1)} = \theta^{(k)} - \alpha(\Delta^{(k)} + \tilde{\Delta}^{(k)})$

// Periodic update of target network

Every C steps reset $\hat{Q} = Q$, i.e., set $\theta^p = \theta$

Return $\theta^{(k+1)}$

Output: Unified multi-agent bit-error robust policy θ

MulBERRY Framework

Start: Initialize swarm autonomy model

Agent error-aware learning; Server-agent communication

Swarm error-aware learning

Goal: learn **robust** swarm autonomy model

Algorithm 2 MulBERRY

Initialization: number of agent n, communication interval CI, smoothing average threshold δ^k. For each agent, initialize action-value function Q with policy θ_i and target action-value function $\hat{Q}$ with policy $\theta^p = \theta$

for *time step* $k = 1$ *to* T **do**

 // Agents conduct bit-flip robust learning at each step

 for *each agent* i **in parallel do**

  Update $\theta_i^k \leftarrow$ `BitFlipLearning` $\left(i, \theta_i^{(k-1)}\right)$

 end for

 // Agents communicate with server at every CI steps

 if k *mod* $CI = 0$ **then**

  Each agent i sends policy θ_i^{k-} to server

  Server calculates smoothing average parameters:

  $\alpha^k = \frac{1}{n}\max(1, \frac{(1-n)k}{\delta^k} + n)$, $\beta^k = \frac{1-\alpha^k}{n-1}$

  for *each agent* i **do**

   Server sends its updated policy θ_i^{k+} back to agent i:

   $\theta_i^{k+} = \alpha^k \theta_i^{k-} + \beta^k \sum_{i \neq j} \theta_j^{k-}$

Function: `BitFlipLearning` $(i, \theta^{(k)})$

 Given state s_k, take action a_k based on Q (ϵ-greedy)

 Obtain reward r_k and reach new state s_{k+1}

 Store transition (s_k, a_k, r_k, s_{k+1}) in D

 // Experience replay

 Sample a mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{b=1}^{B}$ from D

 // Clean training pass

 Set $y_j = r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta^{p(k)})$

 $\Delta^{(k)} = \nabla_\theta \sum_{b=1}^{B} (Q(s_j, a_j; \theta^{(k)}) - y_j)^2$

 // Perturbed training pass, inject bit errors at rate p

 $\tilde{\theta}^{(k)} = BErr_p(\theta^{(k)})$

 Set $\tilde{y_j} = (r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \tilde{\theta}^{p(k)}))$

 $\tilde{\Delta}^{(k)} = \nabla_\theta \sum_{b=1}^{B} (Q(s_j, a_j; \tilde{\theta}^{(k)}) - \tilde{y_j})^2$

 // Average gradients and update w.r.t θ

 $\theta^{(k+1)} = \theta^{(k)} - \alpha(\Delta^{(k)} + \tilde{\Delta}^{(k)})$

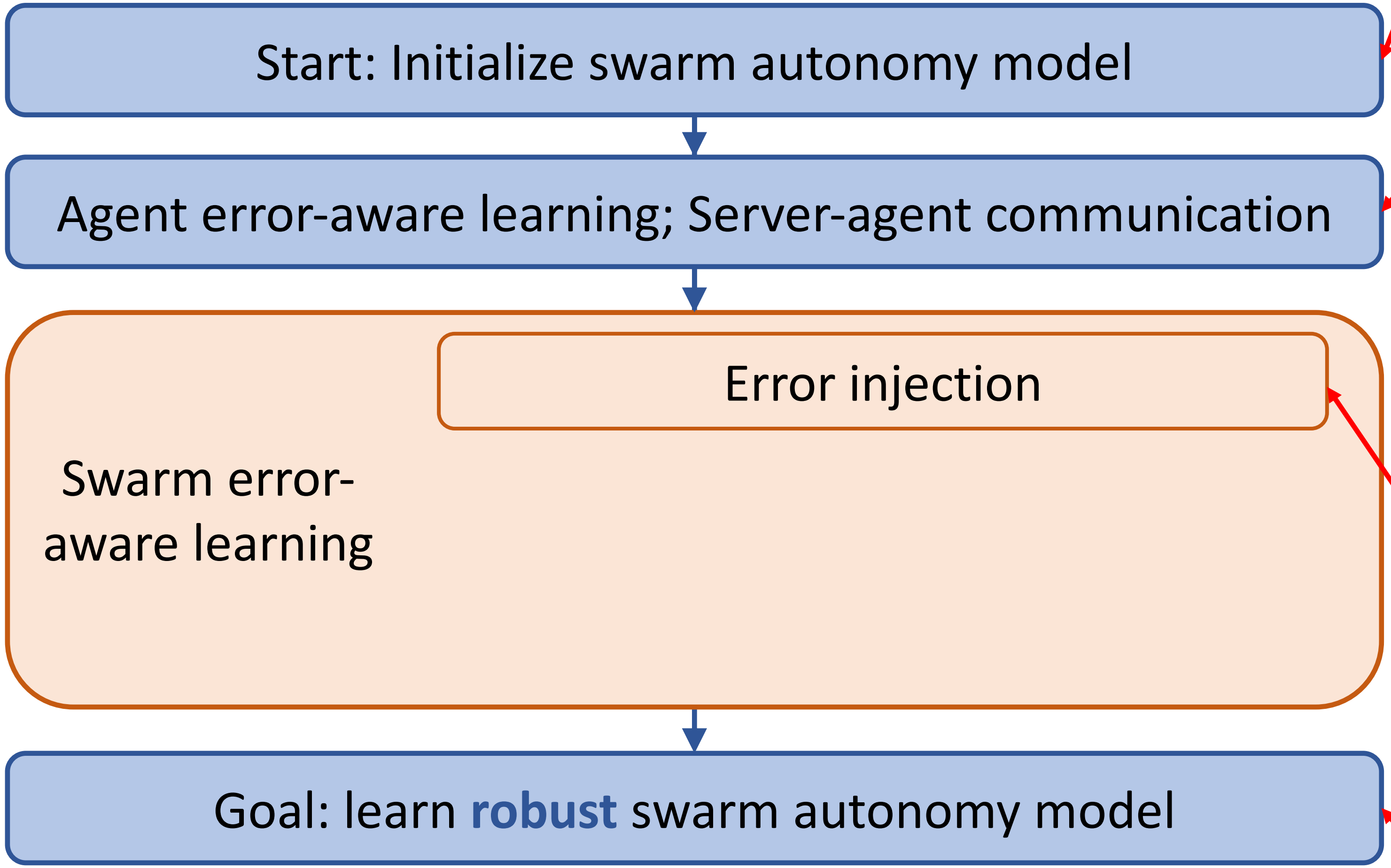
 // Periodic update of target network

 Every C steps reset $\hat{Q} = Q$, i.e., set $\theta^p = \theta$

Return $\theta^{(k+1)}$

Output: Unified multi-agent bit-error robust policy θ

MulBERRY Framework



Algorithm 2 MulBERRY

Initialization: number of agent n, communication interval CI, smoothing average threshold δ^k. For each agent, initialize action-value function Q with policy θ_i and target action-value function $\hat{Q}$ with policy $\theta^p = \theta$

for *time step* $k = 1$ *to* T **do**

  // Agents conduct bit-flip robust learning at each step

  for *each agent* i **in parallel do**

    Update $\theta_i^k \leftarrow \texttt{BitFlipLearning}\left(i, \theta_i^{(k-1)}\right)$

  end for

  // Agents communicate with server at every CI steps

  if $k \ mod \ CI = 0$ **then**

    Each agent i sends policy θ_i^{k-} to server

    Server calculates smoothing average parameters:

    $\alpha^k = \frac{1}{n}\max(1, \frac{(1-n)k}{\delta^k} + n), \ \beta^k = \frac{1-\alpha^k}{n-1}$

    for *each agent* i **do**

      Server sends its updated policy θ_i^{k+} back to agent i:

      $\theta_i^{k+} = \alpha^k \theta_i^{k-} + \beta^k \sum_{i \neq j} \theta_j^{k-}$

Function: $\texttt{BitFlipLearning}\ (i, \theta^{(k)})$

Given state s_k, take action a_k based on Q (ϵ-greedy)

Obtain reward r_k and reach new state s_{k+1}

Store transition (s_k, a_k, r_k, s_{k+1}) in D

// Experience replay

Sample a mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{b=1}^{B}$ from D

// Clean training pass

Set $y_j = r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta^{p(k)})$

$\Delta^{(k)} = \nabla_\theta \sum_{b=1}^{B} (Q(s_j, a_j; \theta^{(k)}) - y_j)^2$

// Perturbed training pass, inject bit errors at rate p

$\tilde{\theta}^{(k)} = BErr_p(\theta^{(k)})$

Set $\tilde{y}_j = (r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \tilde{\theta}^{p(k)}))$

$\tilde{\Delta}^{(k)} = \nabla_\theta \sum_{b=1}^{B} (Q(s_j, a_j; \tilde{\theta}^{(k)}) - \tilde{y}_j)^2$

// Average gradients and update w.r.t θ

$\theta^{(k+1)} = \theta^{(k)} - \alpha(\Delta^{(k)} + \tilde{\Delta}^{(k)})$

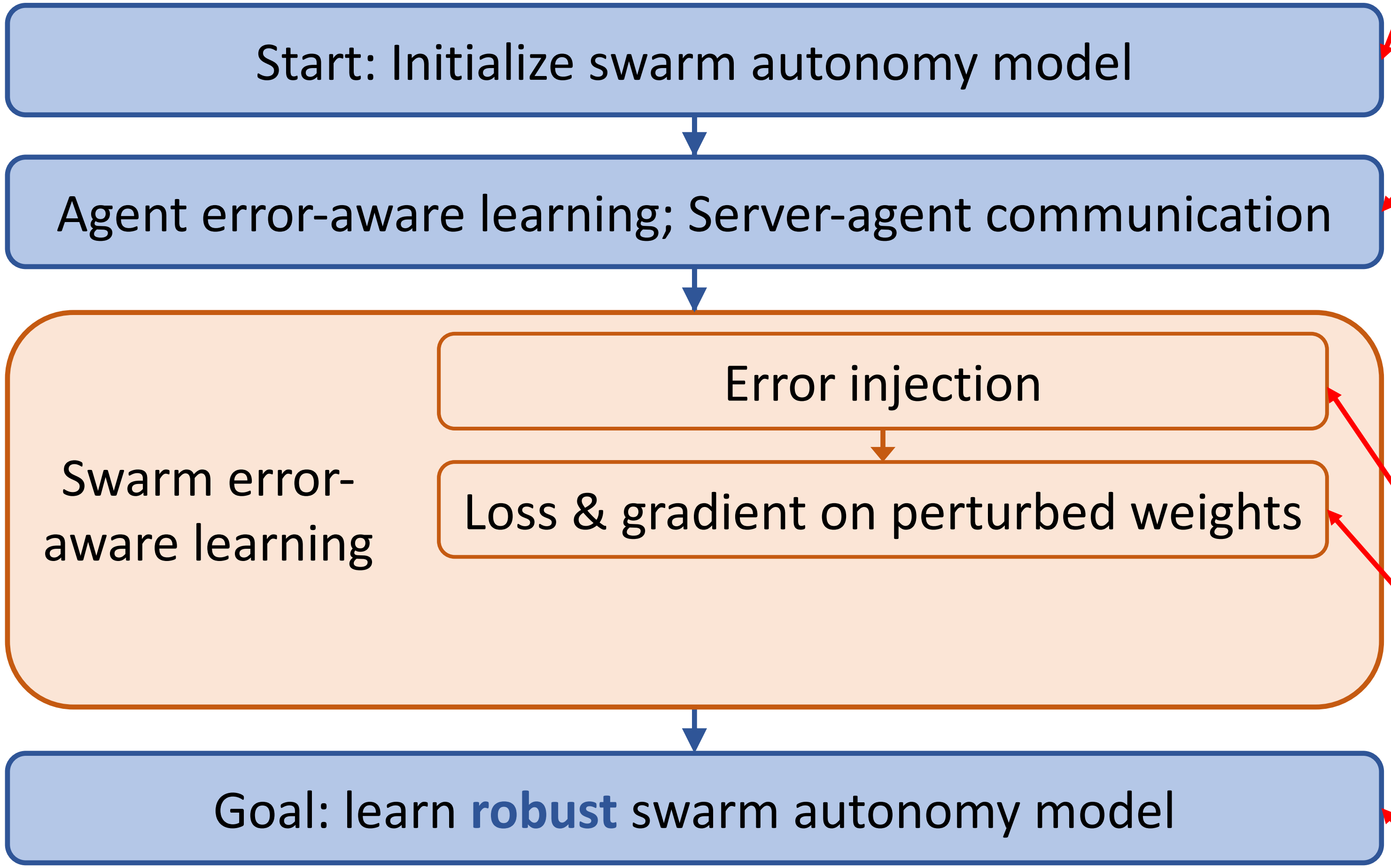
// Periodic update of target network

Every C steps reset $\hat{Q} = Q$, i.e., set $\theta^p = \theta$

Return $\theta^{(k+1)}$

Output: Unified multi-agent bit-error robust policy θ

MulBERRY Framework



Algorithm 2 MulBERRY

Initialization: number of agent n, communication interval CI, smoothing average threshold δ^k. For each agent, initialize action-value function Q with policy θ_i and target action-value function $\hat{Q}$ with policy $\theta^p = \theta$

for *time step* $k = 1$ *to* T **do**

// Agents conduct bit-flip robust learning at each step

for *each agent* i **in parallel do**

Update $\theta_i^k \leftarrow$ `BitFlipLearning` $\left(i, \theta_i^{(k-1)}\right)$

end for

// Agents communicate with server at every CI steps

if k *mod* $CI = 0$ **then**

Each agent i sends policy θ_i^{k-} to server

Server calculates smoothing average parameters:

$\alpha^k = \frac{1}{n}\max(1, \frac{(1-n)k}{\delta^k} + n),\ \beta^k = \frac{1-\alpha^k}{n-1}$

for *each agent* i **do**

Server sends its updated policy θ_i^{k+} back to agent i:

$\theta_i^{k+} = \alpha^k \theta_i^{k-} + \beta^k \sum_{i \neq j} \theta_j^{k-}$

Function: `BitFlipLearning` $(i, \theta^{(k)})$

Given state s_k, take action a_k based on Q (ϵ-greedy)

Obtain reward r_k and reach new state s_{k+1}

Store transition (s_k, a_k, r_k, s_{k+1}) in D

// Experience replay

Sample a mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{b=1}^{B}$ from D

// Clean training pass

Set $y_j = r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta^{p(k)})$

$\Delta^{(k)} = \nabla_\theta \sum_{b=1}^{B} (Q(s_j, a_j; \theta^{(k)}) - y_j)^2$

// Perturbed training pass, inject bit errors at rate p

$\tilde{\theta}^{(k)} = BErr_p(\theta^{(k)})$

Set $\tilde{y}_j = (r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \tilde{\theta}^{p(k)}))$

$\tilde{\Delta}^{(k)} = \nabla_\theta \sum_{b=1}^{B} (Q(s_j, a_j; \tilde{\theta}^{(k)}) - \tilde{y}_j)^2$

// Average gradients and update w.r.t θ

$\theta^{(k+1)} = \theta^{(k)} - \alpha(\Delta^{(k)} + \tilde{\Delta}^{(k)})$

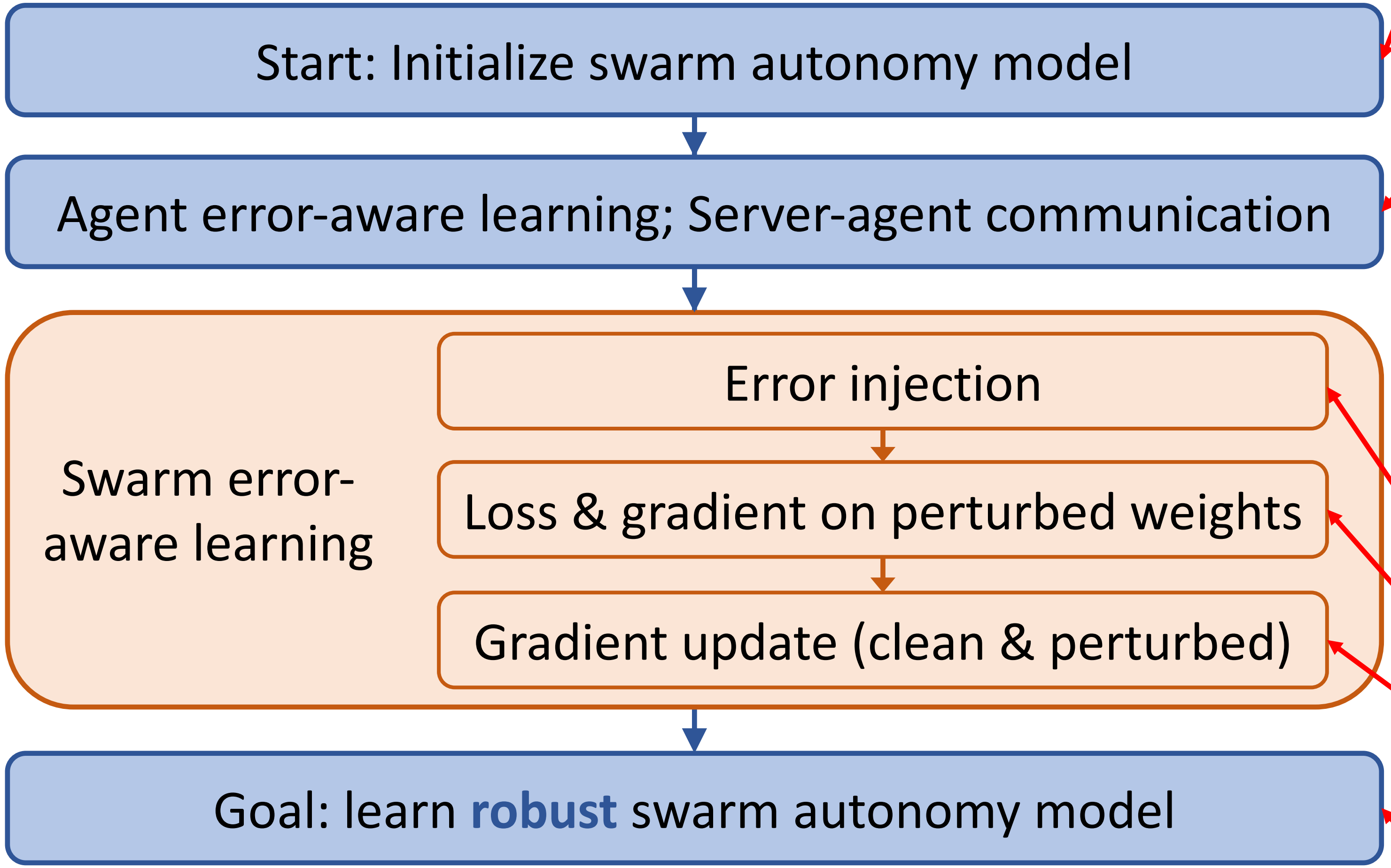
// Periodic update of target network

Every C steps reset $\hat{Q} = Q$, i.e., set $\theta^p = \theta$

Return $\theta^{(k+1)}$

Output: Unified multi-agent bit-error robust policy θ

MulBERRY Framework



Algorithm 2 MulBERRY

Initialization: number of agent n, communication interval CI, smoothing average threshold δ^k. For each agent, initialize action-value function Q with policy θ_i and target action-value function $\hat{Q}$ with policy $\theta^p = \theta$

for *time step* $k = 1$ *to* T **do**

  // Agents conduct bit-flip robust learning at each step

  for *each agent* i **in parallel do**

    Update $\theta_i^k \leftarrow$ `BitFlipLearning` $\left(i, \theta_i^{(k-1)}\right)$

  end for

  // Agents communicate with server at every CI steps

  if k *mod* $CI = 0$ **then**

    Each agent i sends policy θ_i^{k-} to server

    Server calculates smoothing average parameters:

    $\alpha^k = \frac{1}{n} \max(1, \frac{(1-n)k}{\delta^k} + n), \ \beta^k = \frac{1-\alpha^k}{n-1}$

    for *each agent* i **do**

      Server sends its updated policy θ_i^{k+} back to agent i:

      $\theta_i^{k+} = \alpha^k \theta_i^{k-} + \beta^k \sum_{i \neq j} \theta_j^{k-}$

Function: `BitFlipLearning` $(i, \theta^{(k)})$

Given state s_k, take action a_k based on Q (ϵ-greedy)

Obtain reward r_k and reach new state s_{k+1}

Store transition (s_k, a_k, r_k, s_{k+1}) in D

// Experience replay

Sample a mini-batch $\{(s_j, a_j, r_j, s_{j+1})\}_{b=1}^B$ from D

// Clean training pass

Set $y_j = r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta^{p(k)})$

$\Delta^{(k)} = \nabla_\theta \sum_{b=1}^B (Q(s_j, a_j; \theta^{(k)}) - y_j)^2$

// Perturbed training pass, inject bit errors at rate p

$\tilde{\theta}^{(k)} = BErr_p(\theta^{(k)})$

Set $\tilde{y}_j = (r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \tilde{\theta}^{p(k)}))$

$\tilde{\Delta}^{(k)} = \nabla_\theta \sum_{b=1}^B (Q(s_j, a_j; \tilde{\theta}^{(k)}) - \tilde{y}_j)^2$

// Average gradients and update w.r.t θ

$\theta^{(k+1)} = \theta^{(k)} - \alpha(\Delta^{(k)} + \tilde{\Delta}^{(k)})$

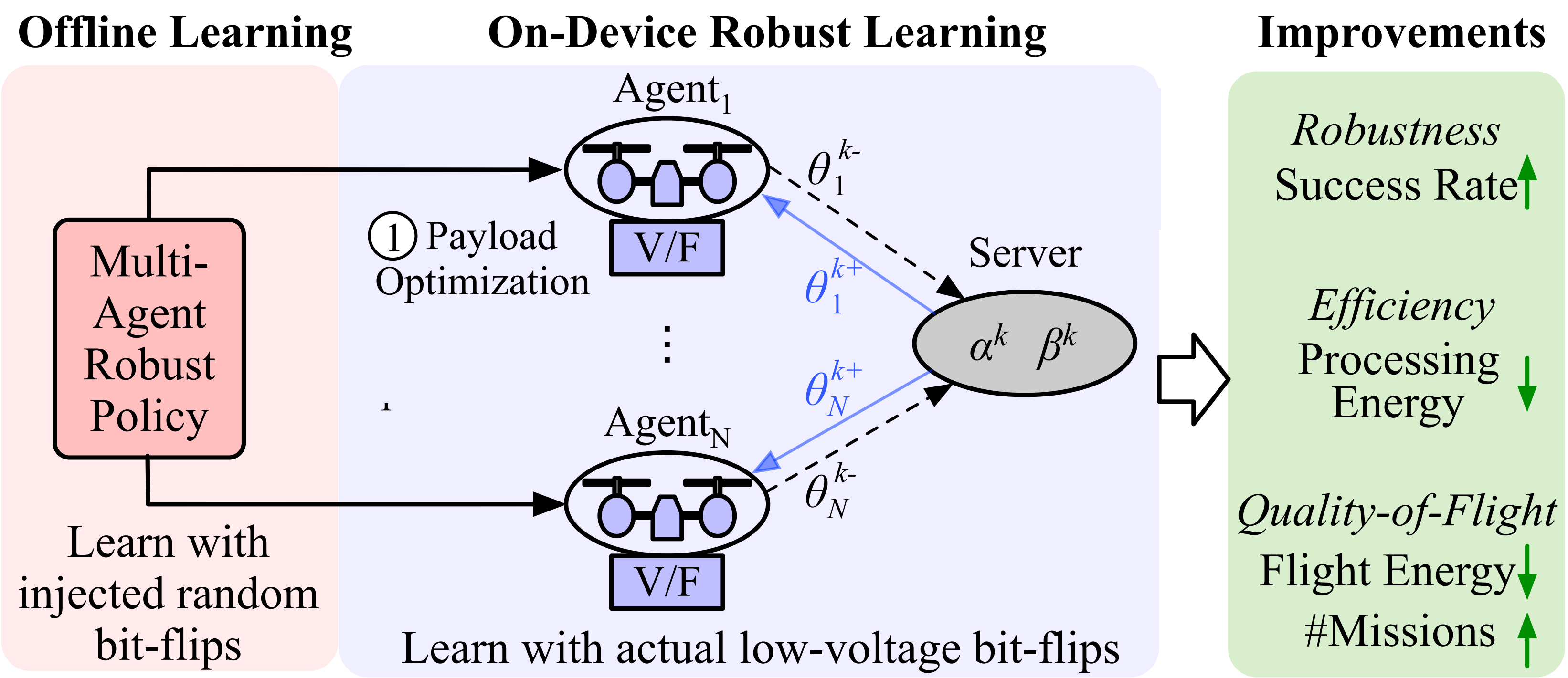
// Periodic update of target network

Every C steps reset $\hat{Q} = Q$, i.e., set $\theta^p = \theta$

Return $\theta^{(k+1)}$

Output: Unified multi-agent bit-error robust policy θ

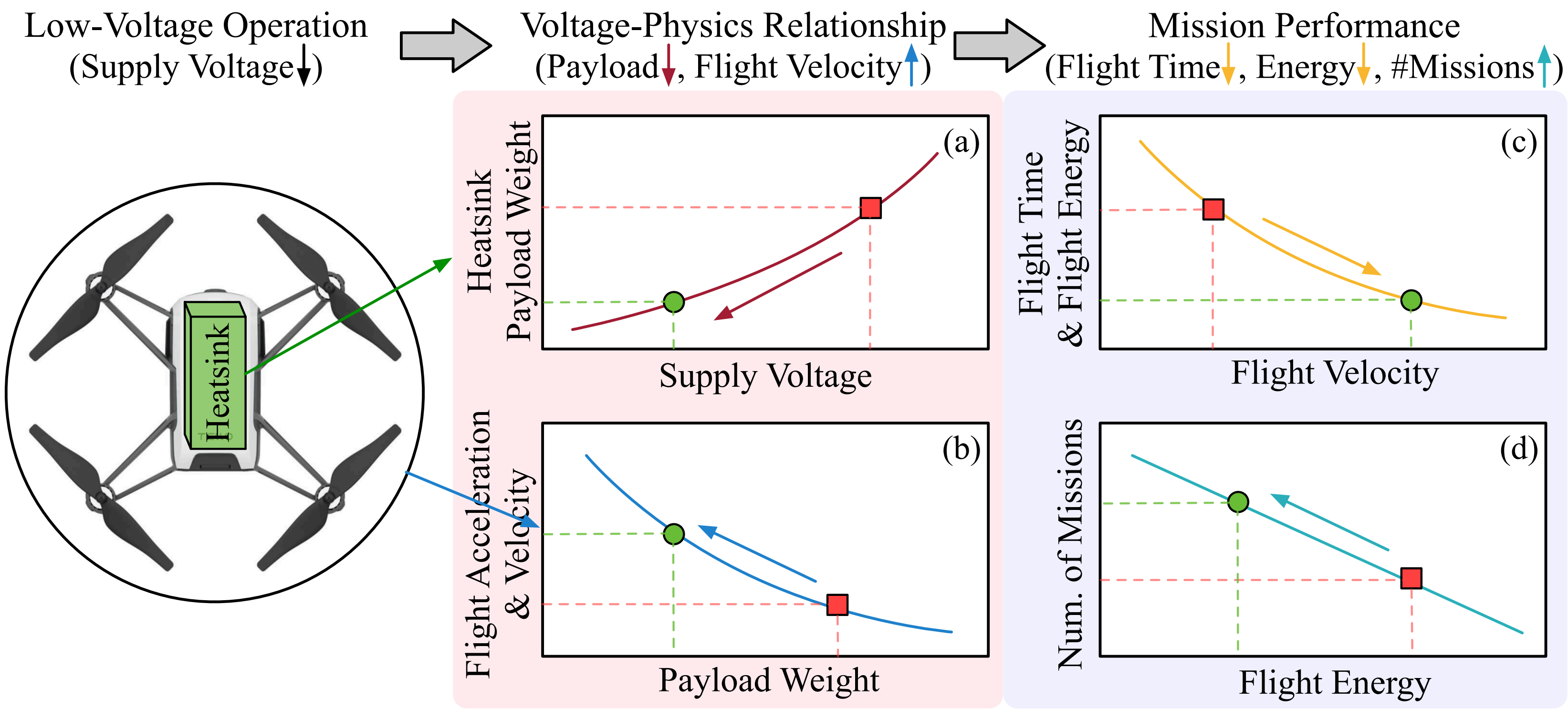
MulBERRY Key Techniques



Two-Stage Swarm Robust Learning

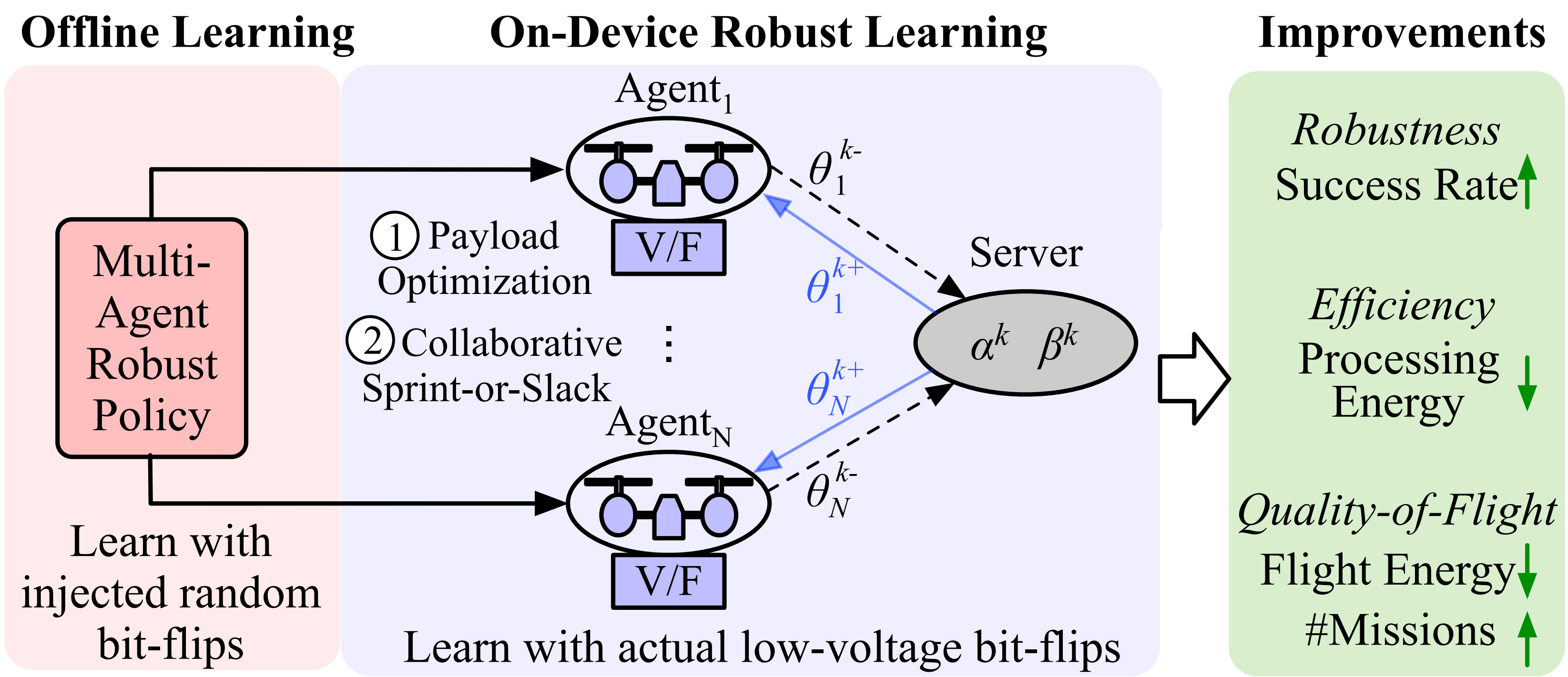
Low-Voltage Payload Optimization

Low-Voltage Payload Optimization



Under low-voltage, MulBERRY reduces drone payload, leading to increased safe flight velocity, thus reducing mission time and energy

MulBERRY Key Techniques



Two-Stage Swarm Robust Learning

Low-Voltage Payload Optimization

Collaborative Sprint-or-Slack Operation

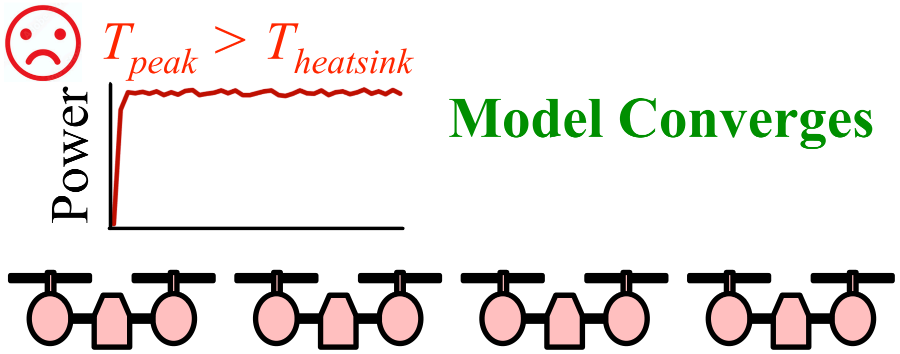
Sprint-or-Slack Operation

- UAV Sprint: operate at nominal voltage (no error, high energy)
- UAV Slack: operate at low voltage (with error, low energy)

Sprint-or-Slack Operation

UAV Sprint: operate at nominal voltage (no error, high energy)

UAV Slack: operate at low voltage (with error, low energy)

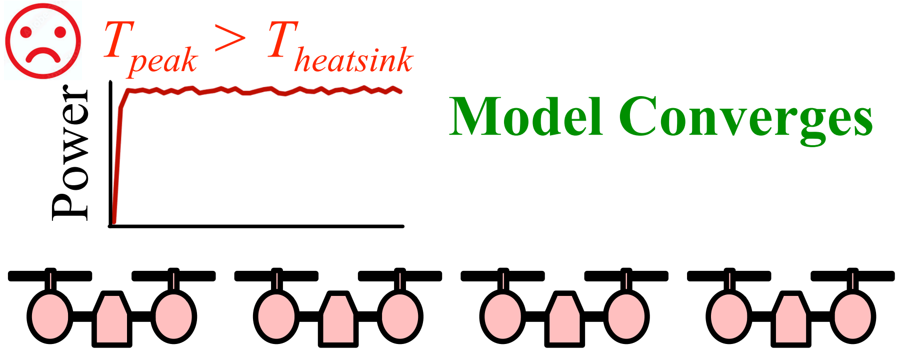


All UAVs Running at V_{sprint}

All UAVs are Sprinting

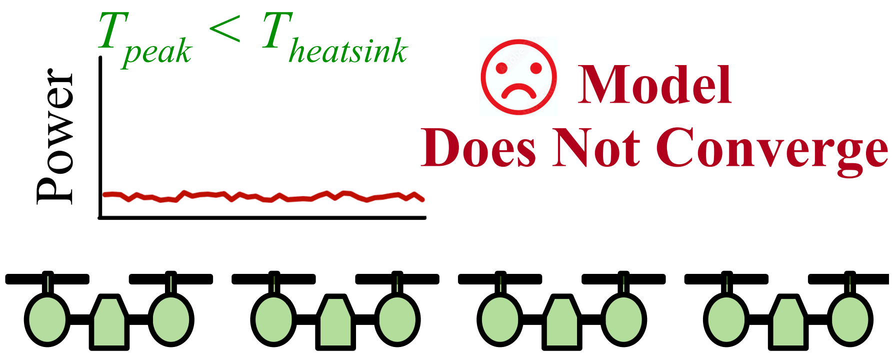
Sprint-or-Slack Operation

- UAV Sprint: operate at nominal voltage (no error, high energy)
- UAV Slack: operate at low voltage (with error, low energy)



All UAVs Running at V_{sprint}

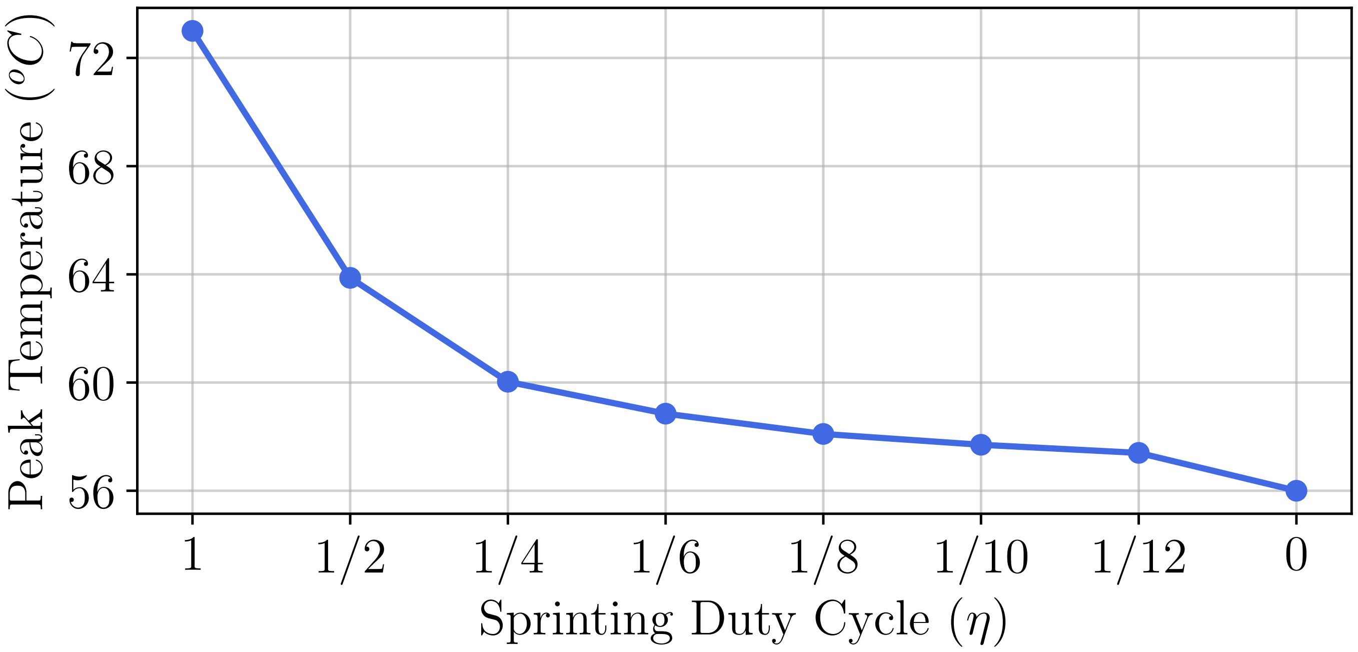
All UAVs are Sprinting



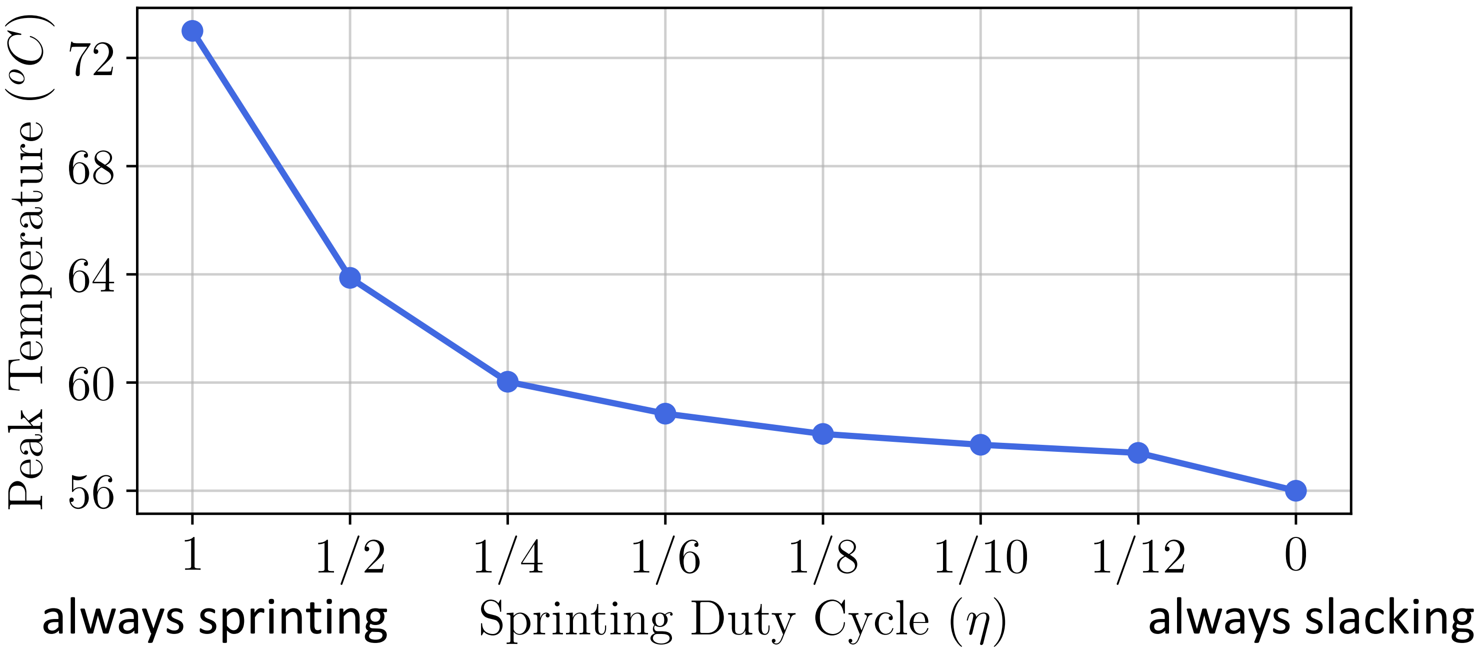
All UAVs Running at V_{slack}

All UAVs are Slacking

Sprint-or-Slack Operation

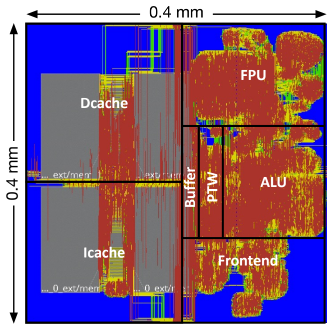


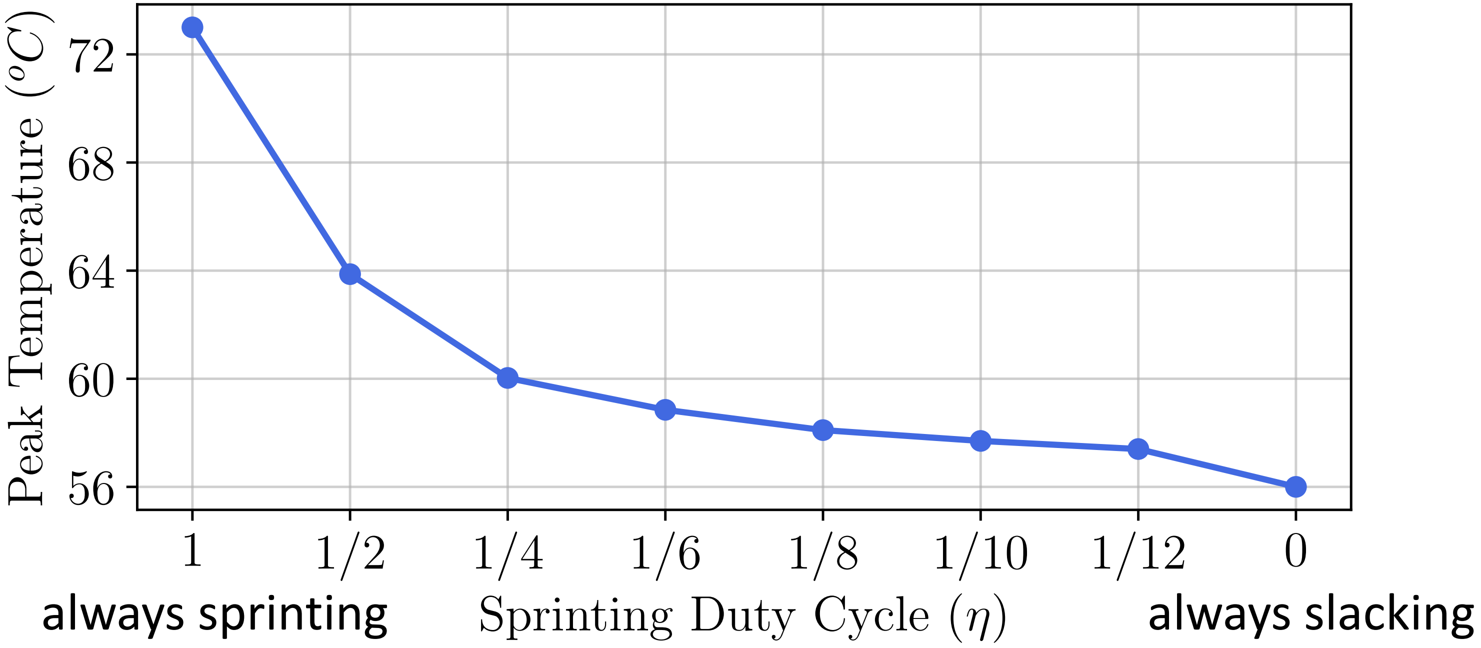
Sprint-or-Slack Operation



Sprinting Duty Cycle (η): fraction of period for which the UAV is sprinting

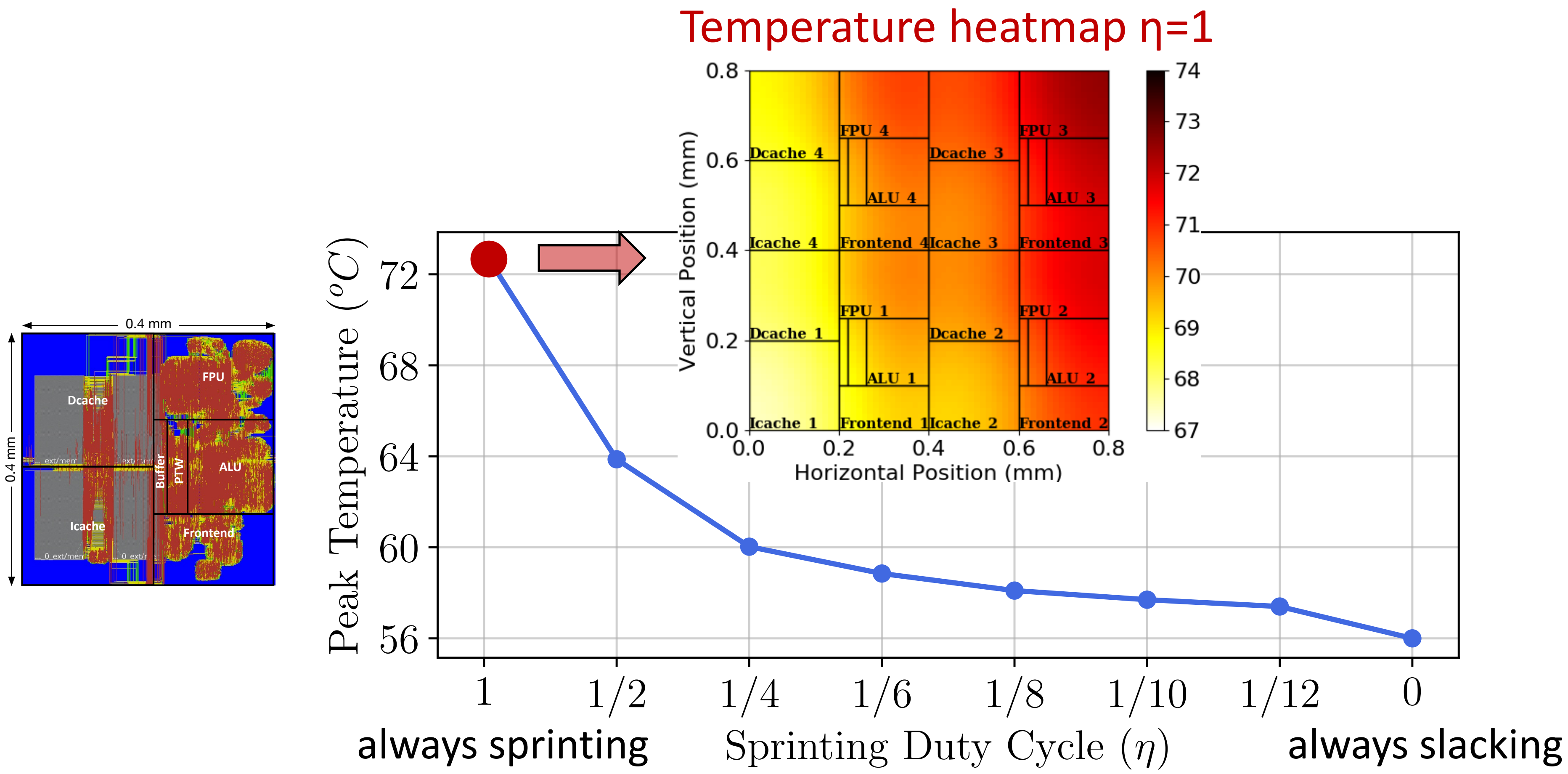
Sprint-or-Slack Operation





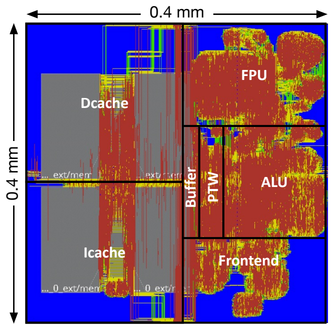
Sprinting Duty Cycle (η): fraction of period for which the UAV is sprinting

Sprint-or-Slack Operation



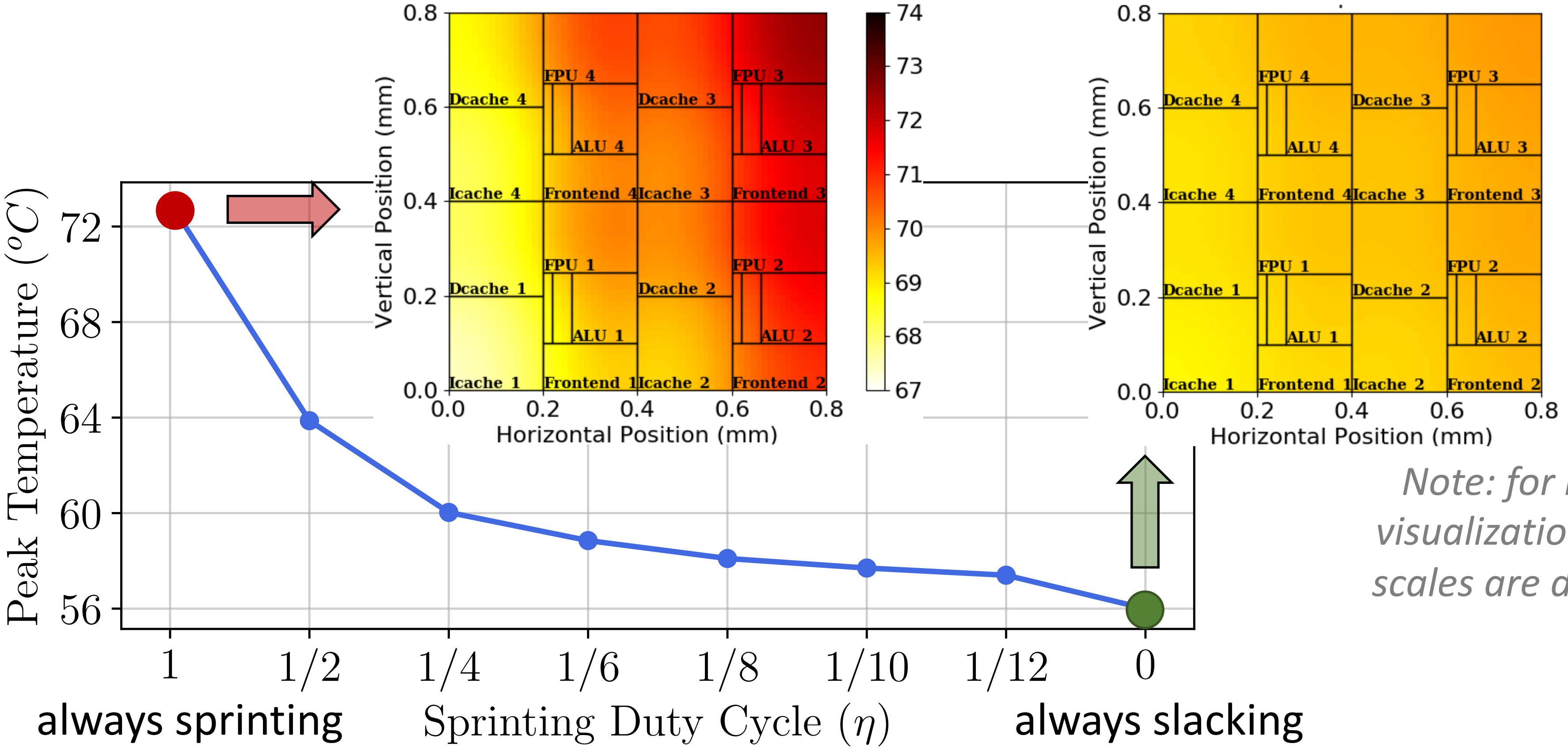
Sprinting Duty Cycle (η): fraction of period for which the UAV is sprinting

Sprint-or-Slack Operation



Temperature heatmap η=1

Temperature heatmap η=0



Note: for better visualization, color scales are different

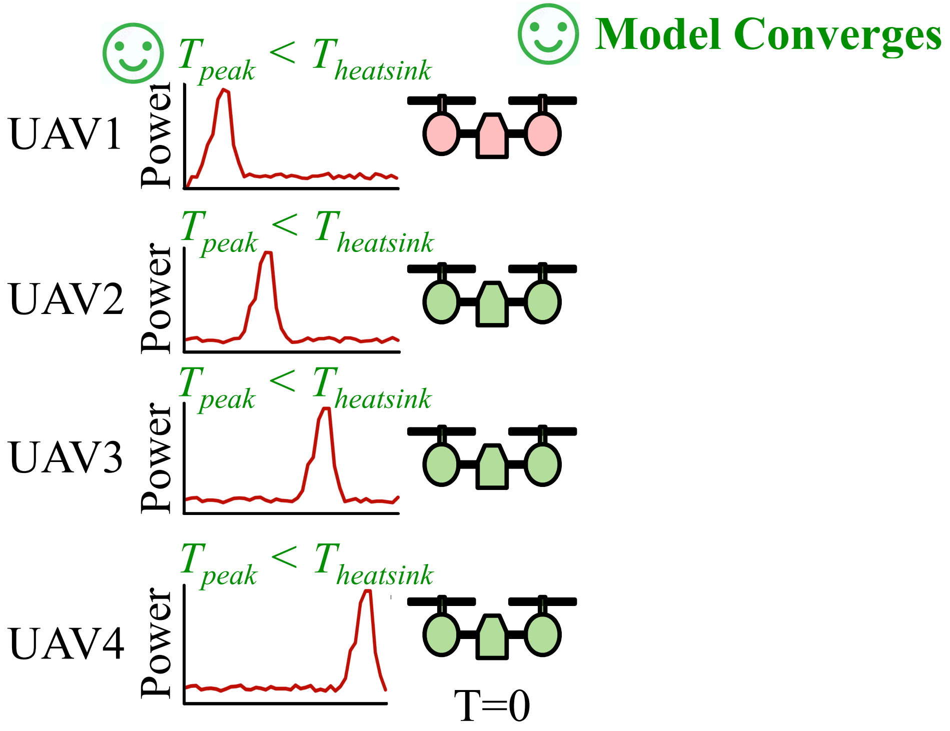
Sprinting Duty Cycle (η): fraction of period for which the UAV is sprinting

Collaborative Sprint-or-Slack Operation

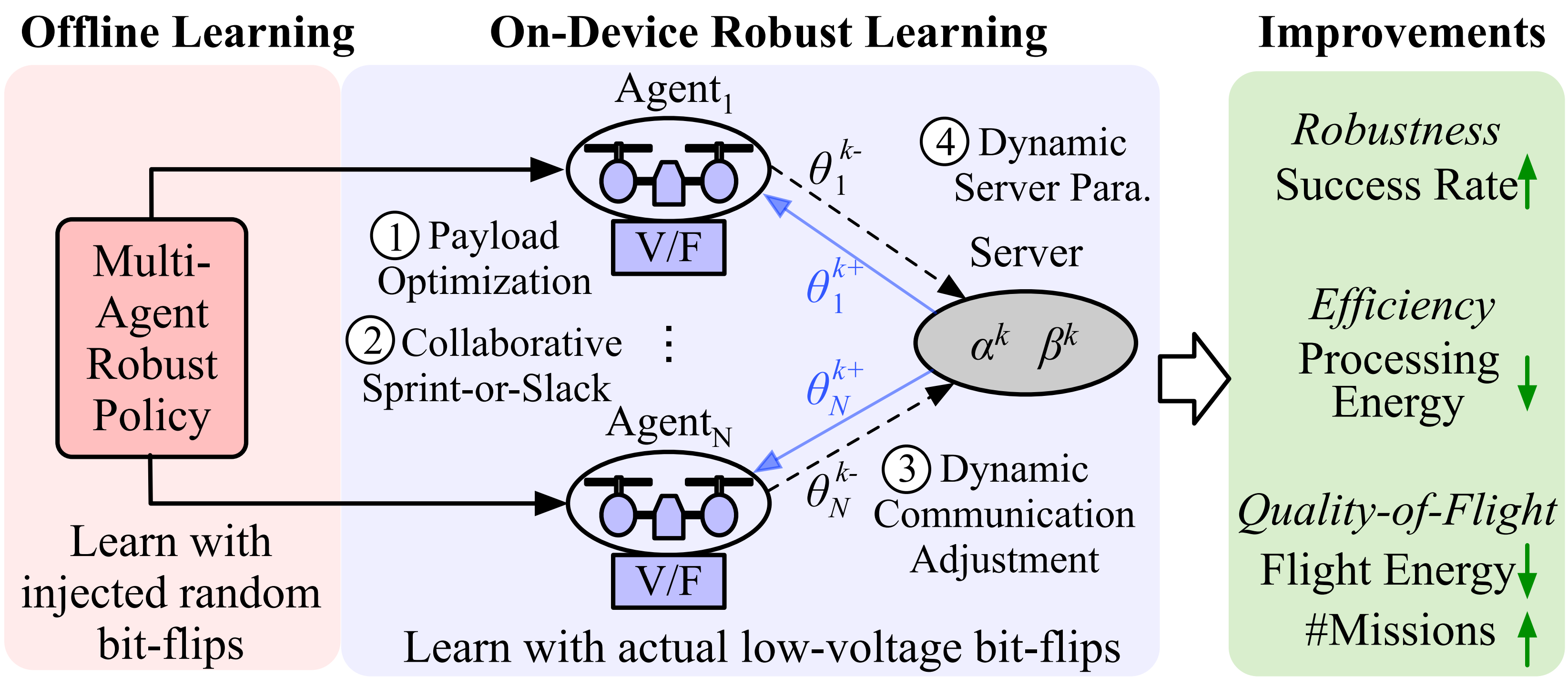
UAV Sprint: operate at nominal voltage (no error, high energy)

UAV Slack: operate at low voltage (with error, low energy)

Collaborative Sprint-or-Slack



MulBERRY Key Techniques



Two-Stage Swarm Robust Learning

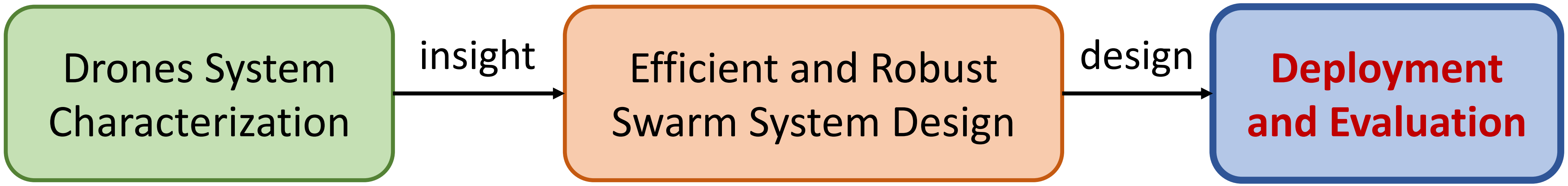
Low-Voltage Payload Optimization

Collaborative Sprint-or-Slack Operation

Adaptive Swarm Knowledge Sharing

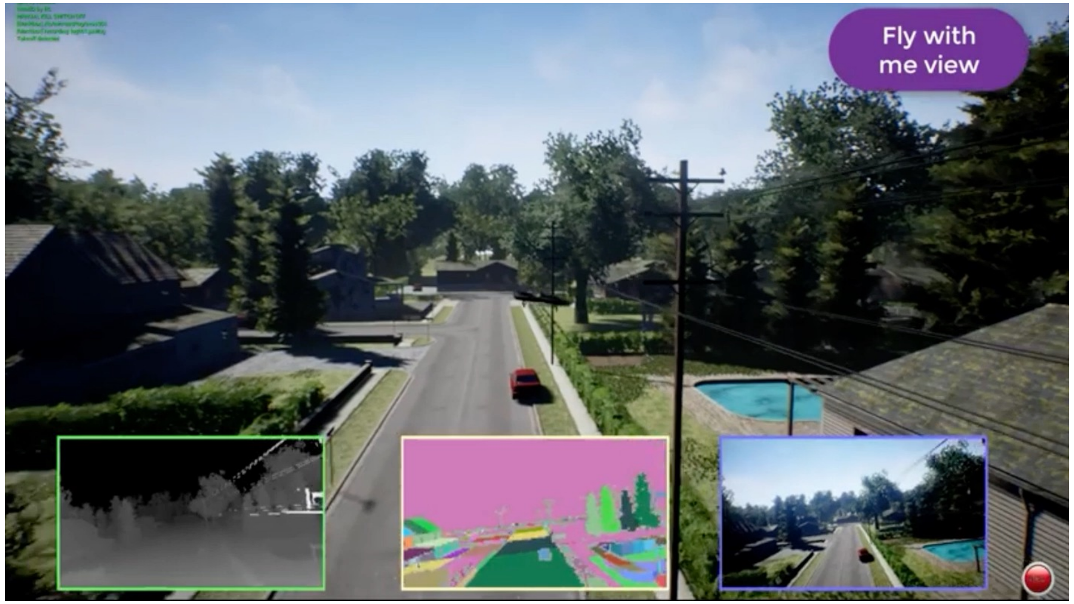
MulBERRY Framework

(MulBERRY: Enabling Bit-Error Robustness for Energy-Efficient Multi-Agent Autonomous Systems)



Swarm UAVs Experimental Setup (Sim/Task)

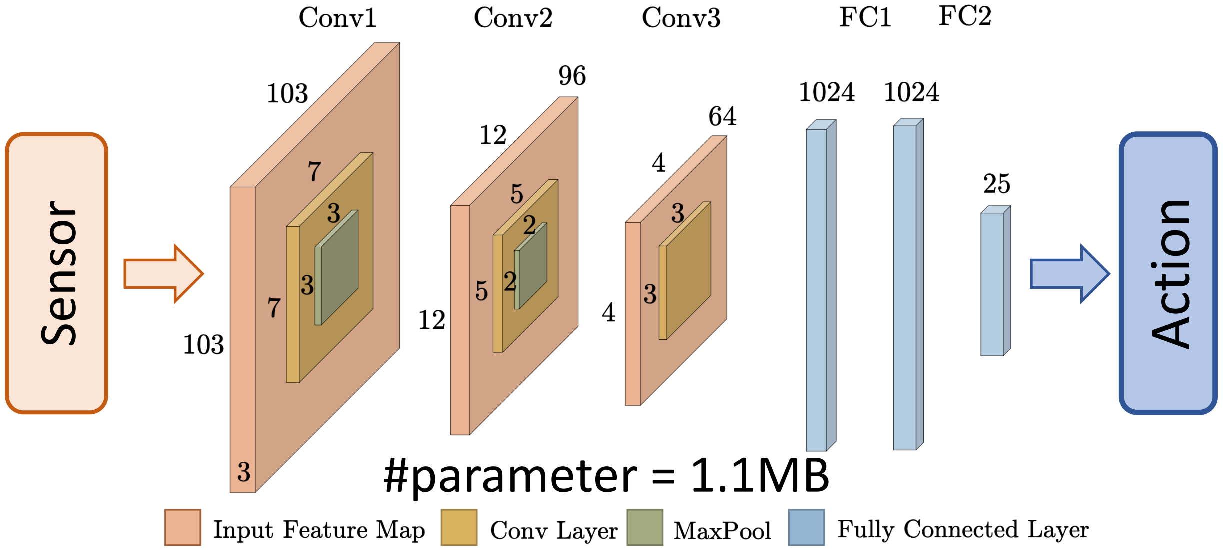
- Simulation Platform:



Unreal Engine + AirSim

(3D realistic environments) (Drone dynamics)

- Task: collaborative package delivery or surveillance
- Policy Architecture of each UAV:



- Swarm size: 4-UAV, 8-UAV, 12-UAV

Swarm UAVs Experimental Setup (UAV Platform)

Bitcraze Crazyflie UAV

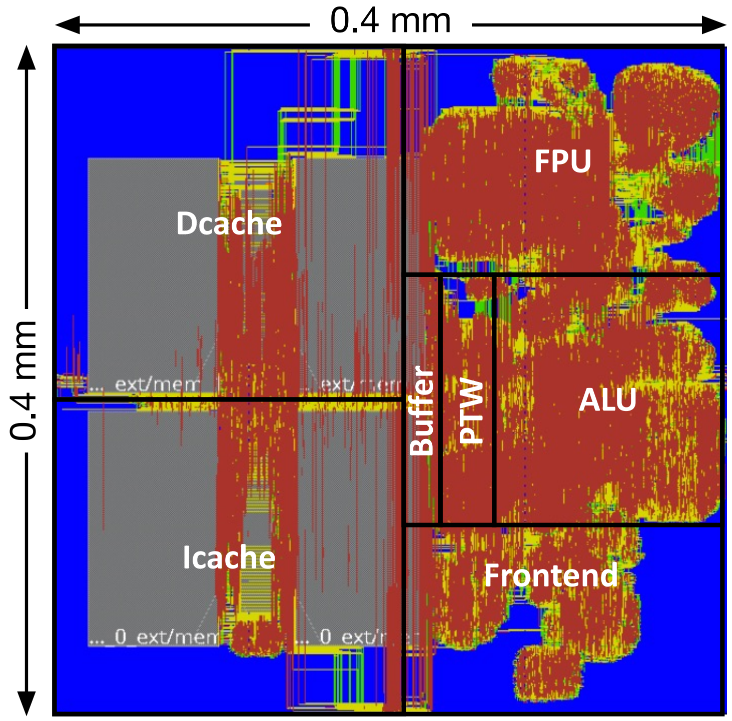
Nano-Drone
27g takeoff weight
15g max payload
250mAh battery

DJI Tello UAV

Micro-Drone
80g takeoff weight
70g max payload
1100mAh battery

Swarm UAVs Experimental Setup (Hardware)

(a)



Layout of one RISC-V Rocket core

(b)

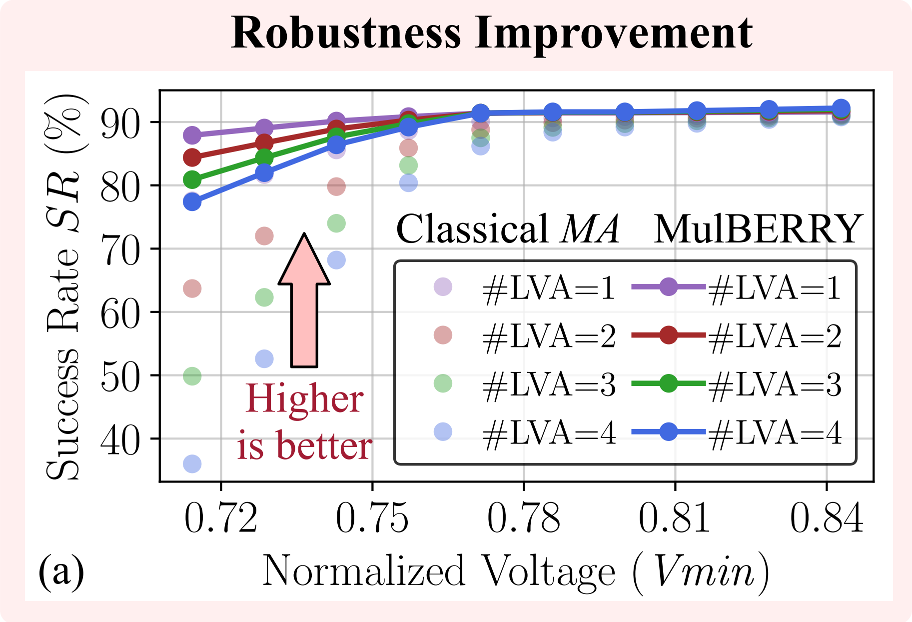
Hardware Configuration Parameters	
Technology	GF 12nm
Core Type	4 x RISC-V Rocket Cores
Cache	16KB 4-way I+D Caches
Routed Core Area	0.4mm x 0.4mm
Voltage Range	0.54V to 1V
Power	117mW to 399mW

Evaluation Metrics

- Compute-level:
  - Processing Energy
- System-level:
  - Avg. flight success rate
  - Avg. flight time
  - Avg. flight energy
  - Avg. #missions

All reported results are averaged from 500 runs

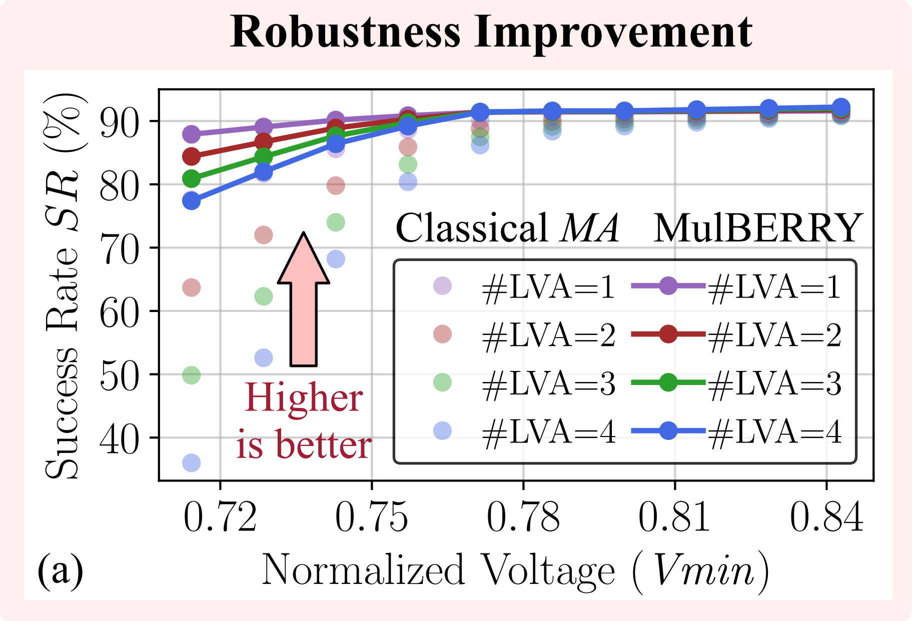
Robustness & Mission Efficiency Improvement

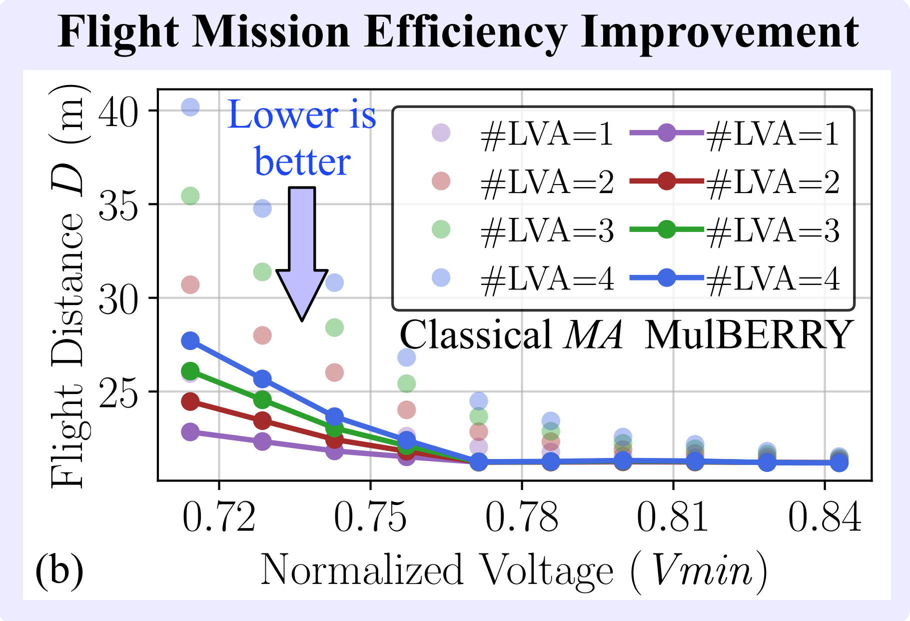


(#LVA: number of low-voltage UAVs)

MulBERRY improves mission robustness under low-voltage operation

Robustness & Mission Efficiency Improvement

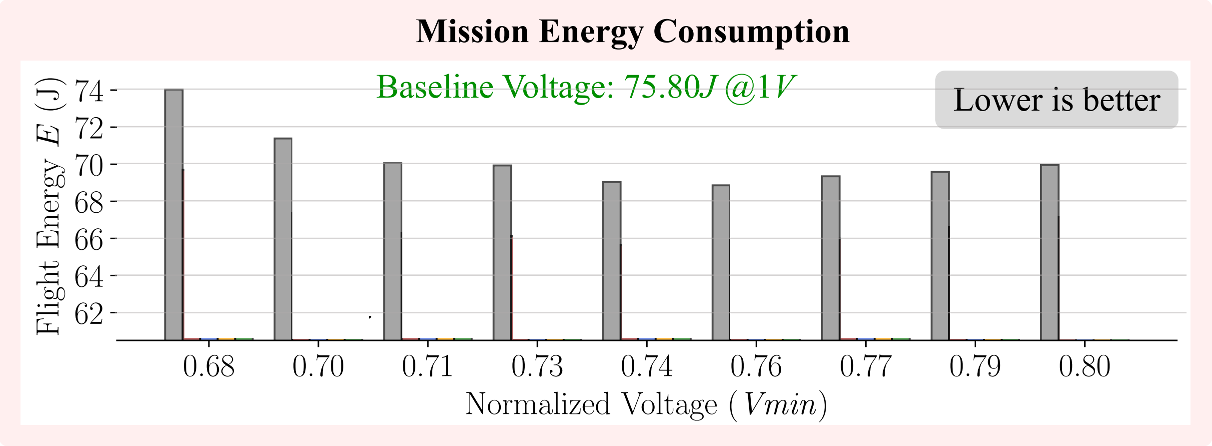


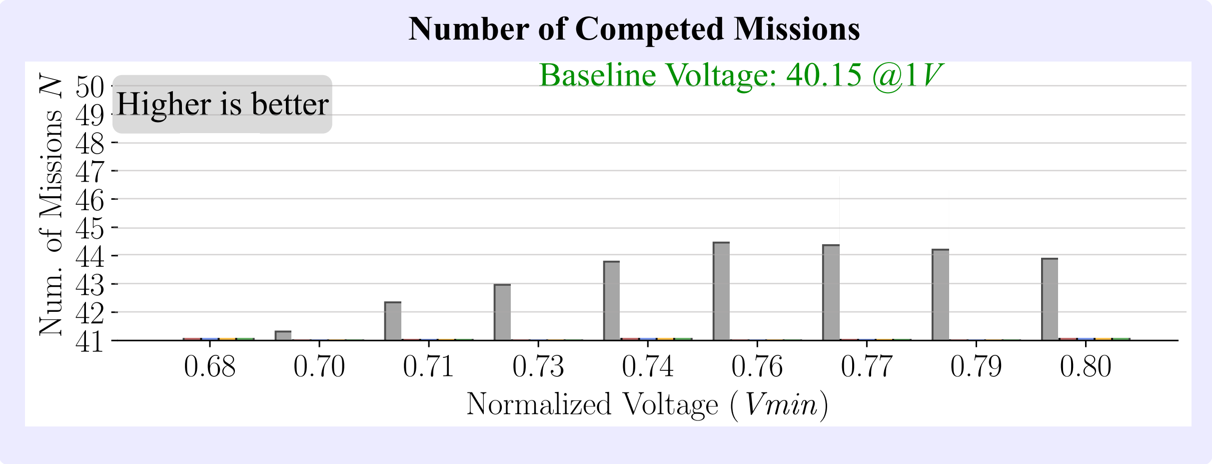


(#LVA: number of low-voltage UAVs)

MulBERRY improves mission robustness under low-voltage operation

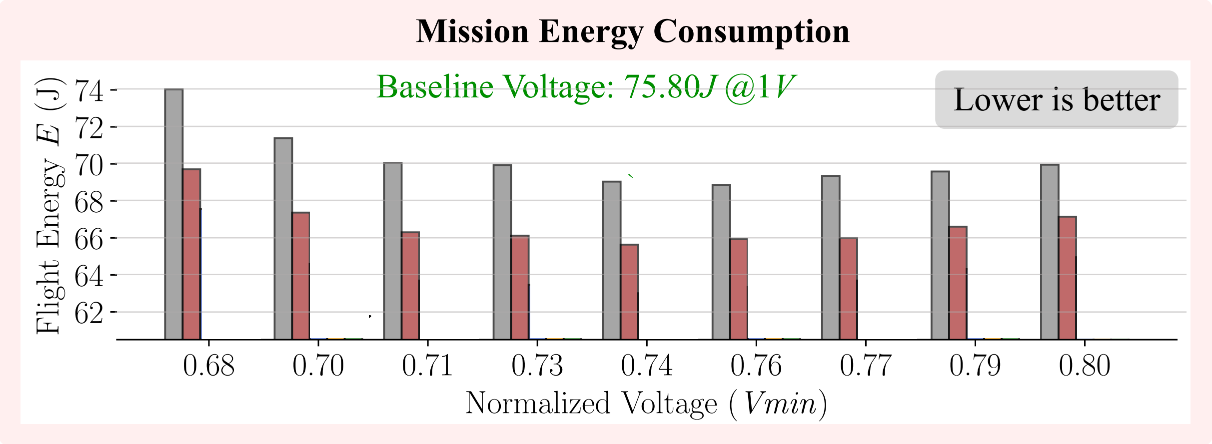
Robustness & Mission Efficiency Improvement

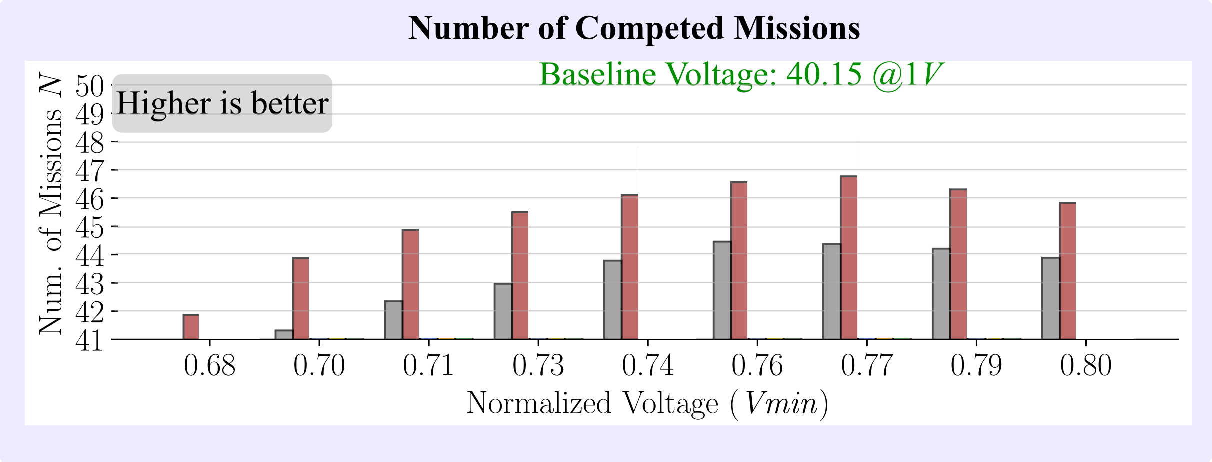




On-Device MulBERRY

Robustness & Mission Efficiency Improvement

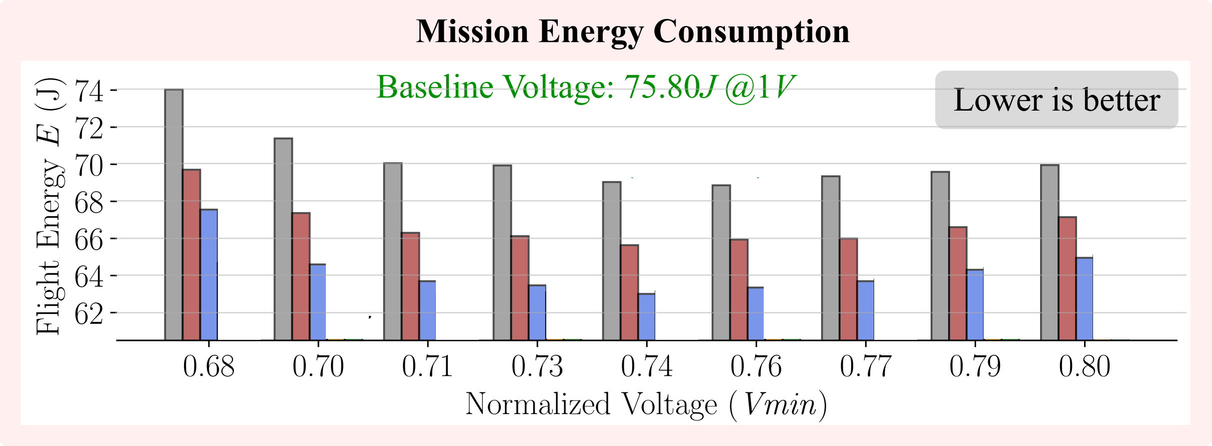


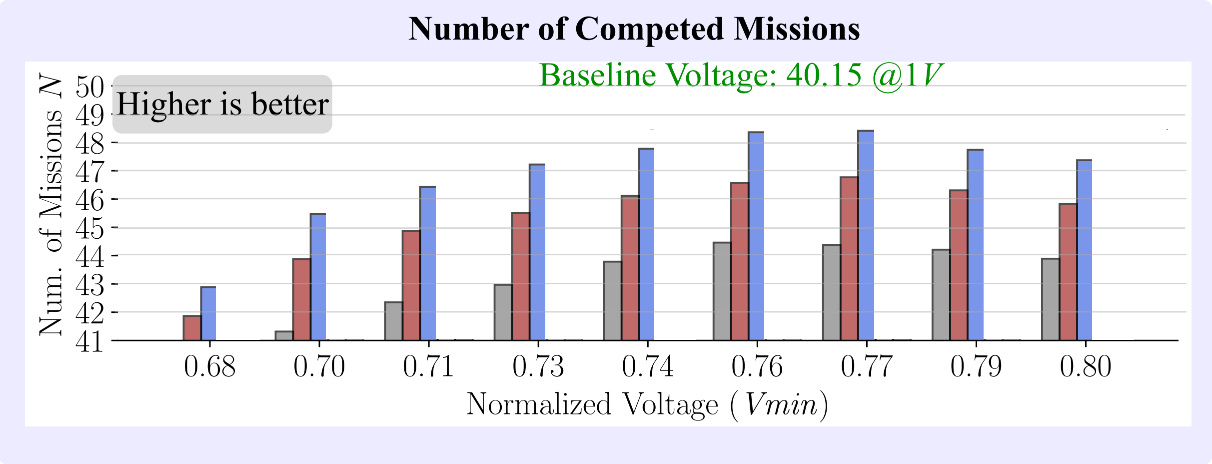


On-Device MulBERRY

+ Payload Optimization

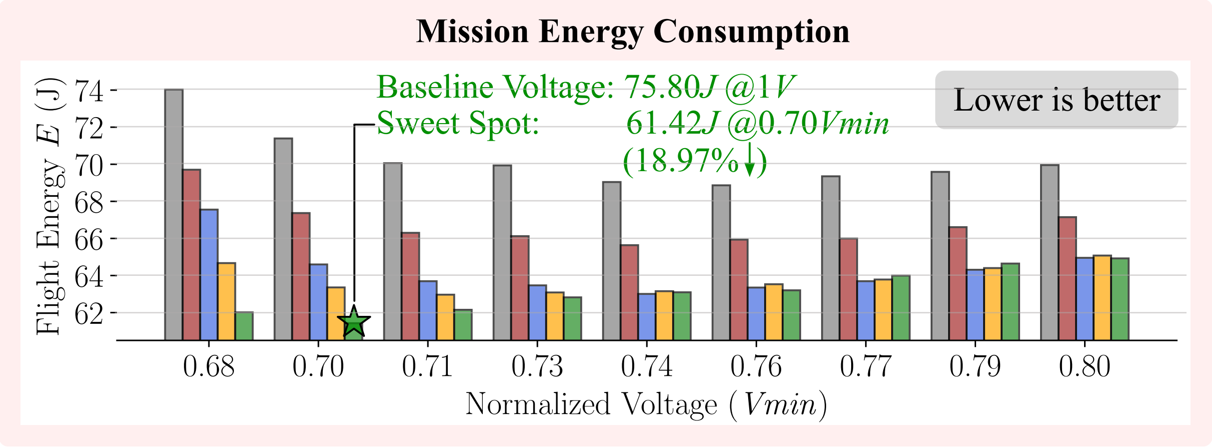
Robustness & Mission Efficiency Improvement

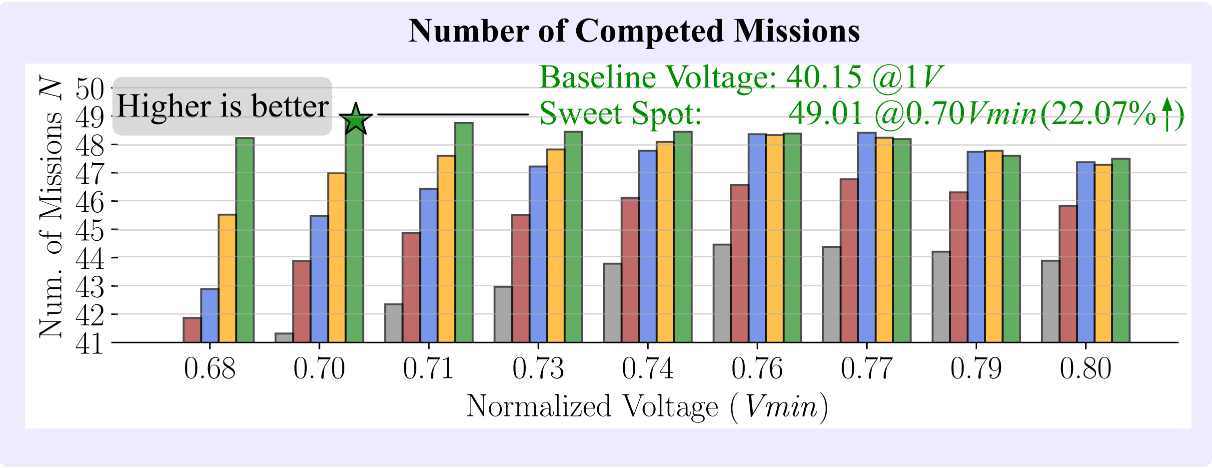




- On-Device MulBERRY
- + Payload Optimization
- + Collaborative Sprint-or-Slack

Robustness & Mission Efficiency Improvement





On-Device MulBERRY

+ Payload Optimization

+ Collaborative Sprint-or-Slack

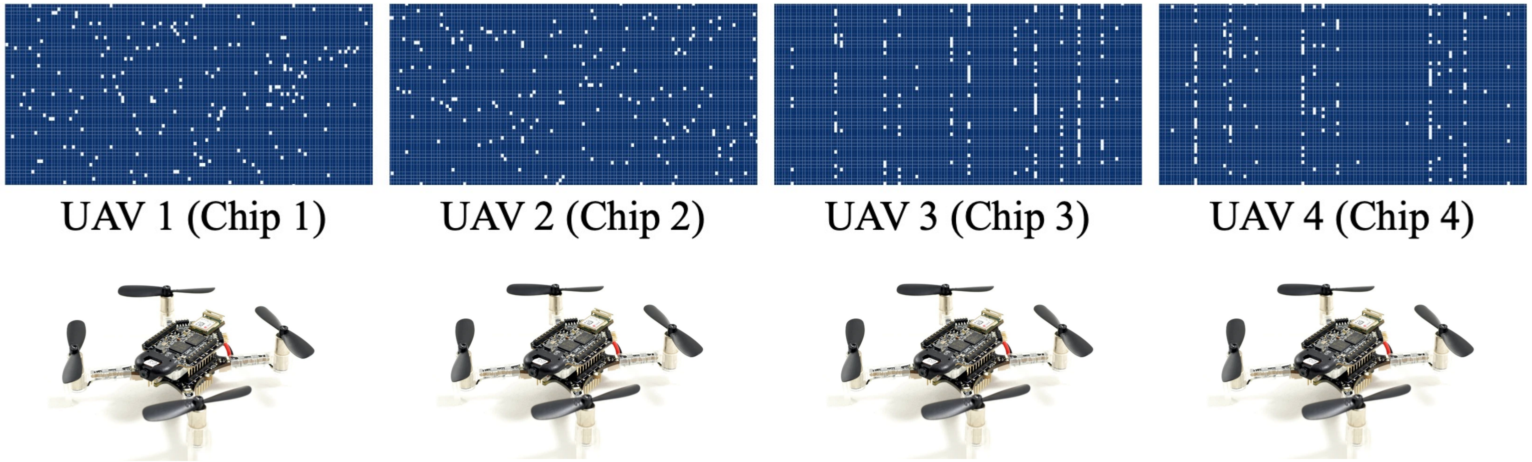
+ Adaptive Communication Interval

+ Adaptive Knowledge Sharing Para.

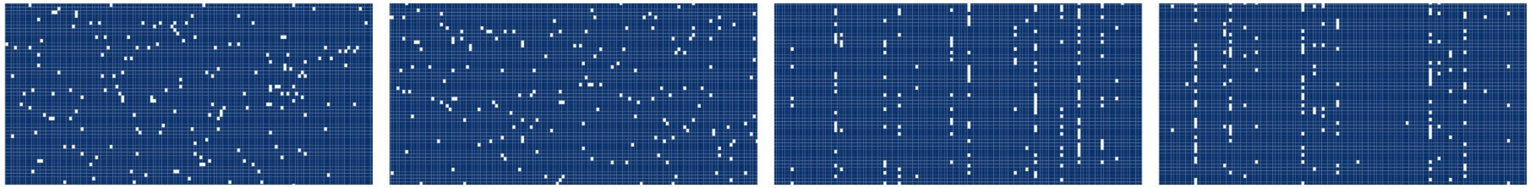
18.97% Less Flight Energy

22.07% More #Completed Missions

Effectiveness Across Voltages and Chips



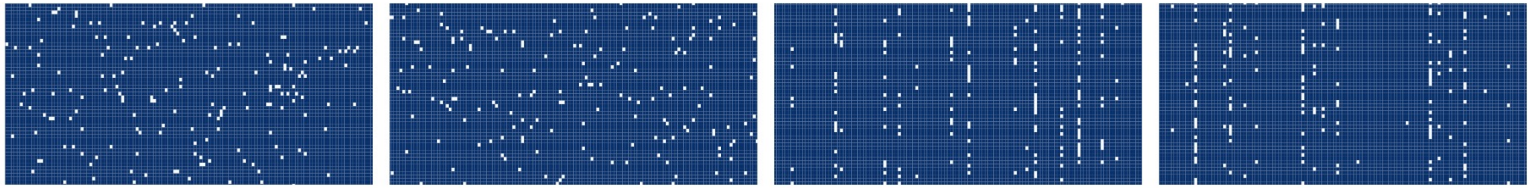
Effectiveness Across Voltages and Chips



UAV 1 (Chip 1) UAV 2 (Chip 2) UAV 3 (Chip 3) UAV 4 (Chip 4)

Voltage / BER (p)	**Metric**	**UAV 1**	**UAV 2**	**UAV 3**	**UAV 4**
Baseline $1V$ (p=0)					
$0.77V_{min}$ / (p=0.025%)	Success Rate (%)				
	Flight Energy (J)				
$0.74V_{min}$ / (p=0.203%)	Success Rate (%)				
	Flight Energy (J)				

Effectiveness Across Voltages and Chips



UAV 1 (Chip 1) UAV 2 (Chip 2) UAV 3 (Chip 3) UAV 4 (Chip 4)

Voltage / BER (p)	**Metric**	**UAV 1**	**UAV 2**	**UAV 3**	**UAV 4**
Baseline $1V$ (p=0)	Success Rate = 91.4%, Flight Energy = 75.80J				
$0.77V_{min}$ / (p=0.025%)	Success Rate (%)	91.6	91.4	90.2	90.6
	Flight Energy (J)	63.90	64.06	66.16	65.47
$0.74V_{min}$ / (p=0.203%)	Success Rate (%)	91.4	91.6	90.4	90.2
	Flight Energy (J)	63.15	62.95	64.37	64.78

MulBERRY is scalable across voltages and chips, and consistently improves efficiency and robustness

Effectiveness Across Environments

Sparse Obstacle

Medium Obstacle

Dense Obstacle

Effectiveness Across Environments

Sparse Obstacle

Medium Obstacle

Dense Obstacle

Environment	Sparse		Medium		Dense	
	Flight Energy (J)	Num. of Missions	Flight Energy (J)	Num. of Missions	Flight Energy (J)	Num. of Missions
Baseline @1V						
MulBERRY (optimal)						

Effectiveness Across Environments


Sparse Obstacle

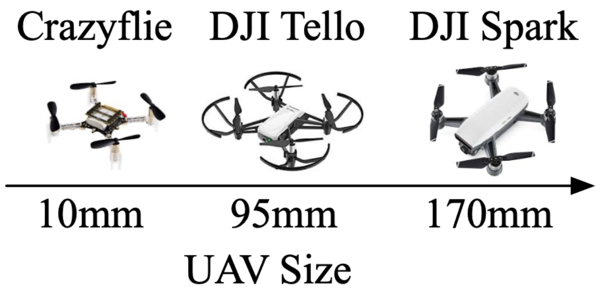

Medium Obstacle


Dense Obstacle

Environment	Sparse		Medium		Dense	
	Flight Energy (J)	Num. of Missions	Flight Energy (J)	Num. of Missions	Flight Energy (J)	Num. of Missions
Baseline @1V	52.41	58.56	75.80	40.15	102.4	28.04
MulBERRY (optimal)	42.02	71.63	61.42	49.01	85.77	33.79
	@0.69V_{min}		@0.70V_{min}		@0.73V_{min}	

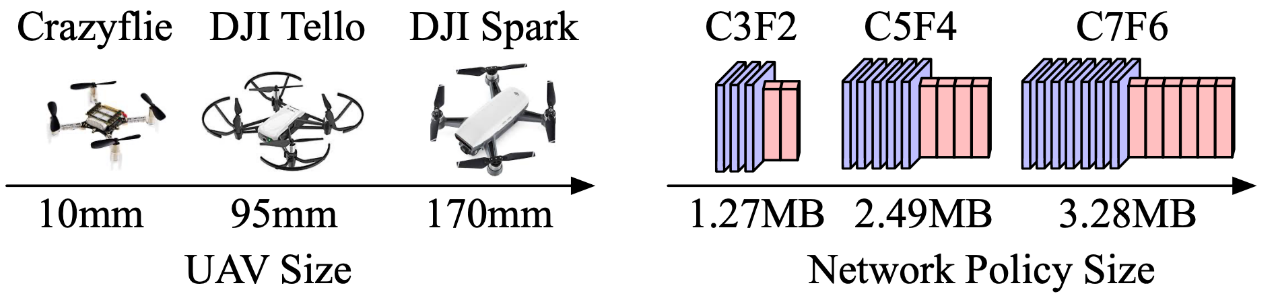
MulBERRY is adaptive across environments, and consistently improves efficiency;
Sparse obstacle environments enable lower operating voltage

Effectiveness Across Drones and Models



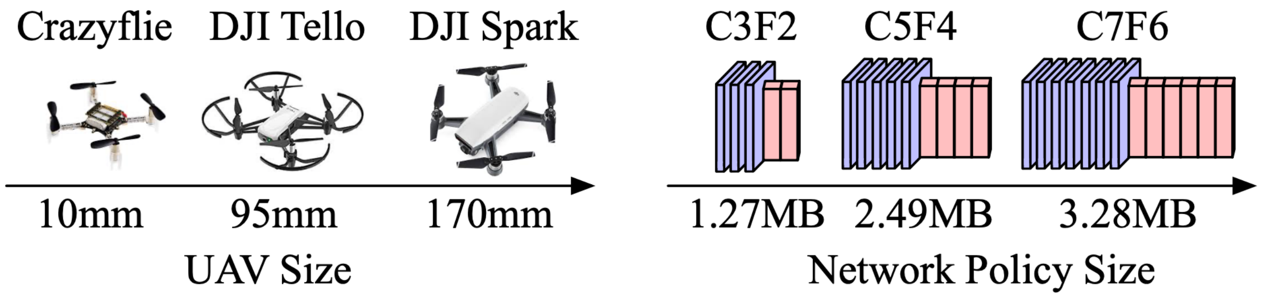
UAV Type
Crazyflie
DJI Tello
DJI Tello
DJI Spark
DJI Spark
DJI Spark

Effectiveness Across Drones and Models



UAV Type	Network Policy
Crazyflie	C3F2
DJI Tello	C3F2
DJI Tello	C5F4
DJI Spark	C3F2
DJI Spark	C5F4
DJI Spark	C7F6

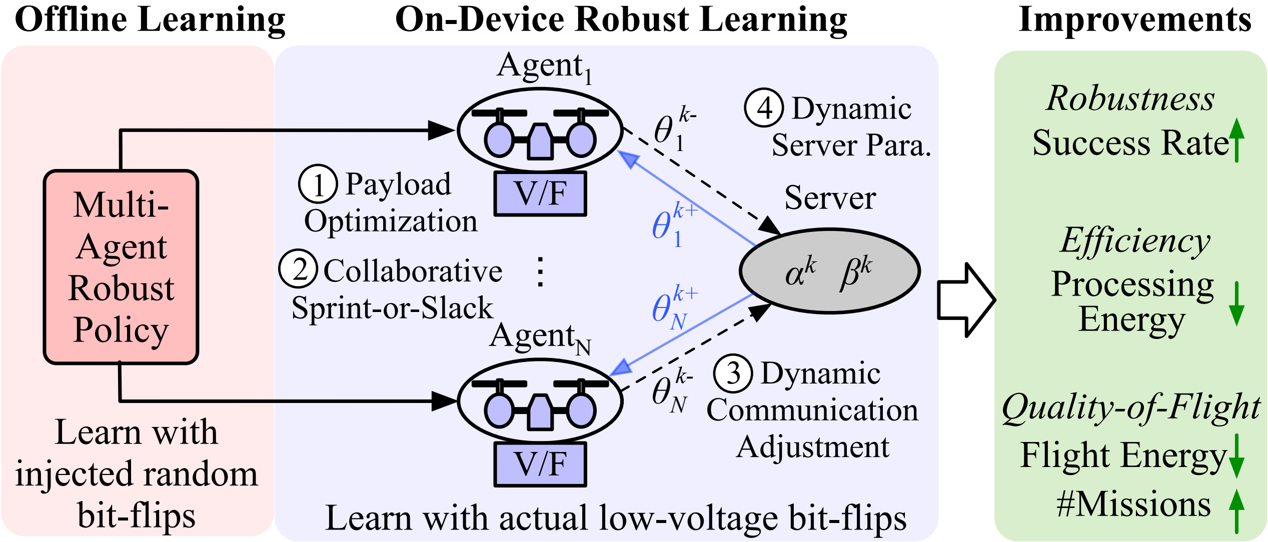
Effectiveness Across Drones and Models



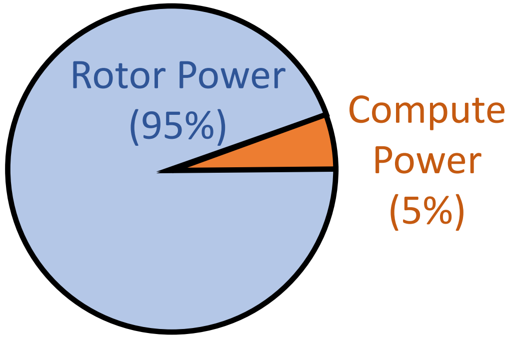
UAV Type	Network Policy	Rotor Power	Compute Power	MulBERRY Flight Energy ↓	MulBERRY #Missions ↑
Crazyflie	C3F2	93.5%	6.5%	18.97%	22.07%
DJI Tello	C3F2	97.4%	2.6%	13.37%	14.16%
DJI Tello	C5F4	95.0%	5.0%	16.04%	17.85%
DJI Spark	C3F2	98.7%	1.3%	6.81%	7.08%
DJI Spark	C5F4	97.5%	2.5%	12.07%	12.92%
DJI Spark	C7F6	96.7%	3.3%	13.88%	14.83%

MulBERRY is adaptive across drones and models, and consistently improves efficiency and robustness; MulBERRY enables more mission energy savings under smaller UAVs and larger models

Summary



Drones System Characterization → insight → Efficient and Robust Swarm System Design → design → Deployment and Evaluation



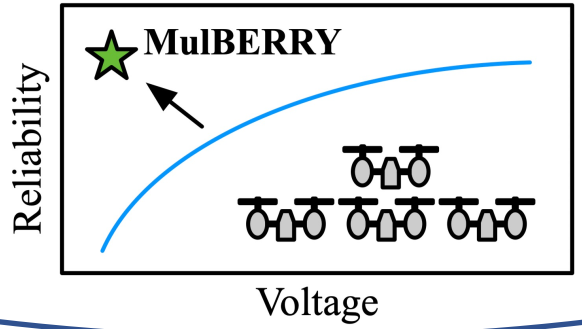
Compute power: small portion, big impact!

Aggressive energy-saving yet computational-resilient

Summary

Reliability

MulBERRY *achieve aggressive energy-savings under low-voltage operation, yet remain computationally-resilient for swarm autonomous systems*



Efficiency

Performance

ASPLOS

Paper Webpage

MulBERRY: Enabling Bit-Error Robustness for Energy-Efficient Multi-Agent Autonomous Systems

Zishen Wan[1], Nandhini Chandramoorthy[2], Karthik Swaminathan[2], Pin-Yu Chen[2], Kshitij Bhardwaj[3], Vijay Janapa Reddi[4], Arijit Raychowdhury[1]

Email: zishenwan@gatech.edu
Web: https://zishenwan.github.io

Georgia Tech IBM Lawrence Livermore National Laboratory HARVARD UNIVERSITY