

# SwarmUpdate: Hierarchical Software Updates and Deep Learning Model Patching for Heterogeneous UAV Swarms

Lin Geng  
20lg5@queensu.ca  
Queen's University  
Kingston, Ontario, Canada

Sidney Givigi  
sidney.givigi@queensu.ca  
Queen's University  
Kingston, Ontario, Canada

Hao Li  
hao.li@queensu.ca  
Queen's University  
Kingston, Ontario, Canada

Bram Adams  
bram.adams@queensu.ca  
Queen's University  
Kingston, Ontario, Canada

## Abstract

Heterogeneous unmanned aerial vehicle (UAV) swarms consist of dozens to hundreds of drones with different roles and varying hardware and software requirements collaborating towards a shared mission. While traditional approaches for synchronized software updates assume swarms to be unstructured and homogeneous, the heterogeneous nature of modern swarms and the emerging need of drones to update their deep learning (perception) models with new objectives or data as a mission unfolds, has made efficient software update methods crucial for swarms to adapt to dynamic environments. To address these challenges, we introduce the SwarmUpdate framework for software updates in heterogeneous UAV swarms, composed of two key components: SwarmSync and SwarmModelPatch. SwarmSync is a hierarchical software update synchronization strategy to distribute a software update to the right subset of drones within a swarm, while SwarmModelPatch is a deep learning model patching method that reduces the size of a (deep learning model) update by only allowing some layers of the model to be updated (freezing the other layers). In this paper, we systematically evaluate the performance of SwarmSync through large-scale simulations in the ARGoS swarm simulator, comparing SwarmSync to auction-based (SOUL) and gossip-based rebroadcasting (Gossip) baselines, and SwarmModelPatch to a non-incremental model patching strategy. Our results show that SwarmSync achieves up to 78.3% faster update convergence compared to SOUL, and up to 47.7% faster convergence compared to Gossip, while maintaining reasonable overhead, making it well-suited for large-scale swarm deployments. Furthermore, our findings show that freezing seven out of eight layers of the model can reduce the update size by 73.3%, speeding up the update process by an average of 72.2% and reducing transmission overhead by an average of 74.3%, at the expense of a drop of 5.1% in overall accuracy (from 72.7% down to 67.6%).

## CCS Concepts

• Software and its engineering; • Computing methodologies  
→ Machine learning;

## Keywords

Heterogeneous UAV swarms, software update, model patching

## 1 Introduction

Autonomous unmanned aerial vehicle (UAV) swarms have emerged as a promising technology for accomplishing complex missions without relying on a central server [42]. The last decade, research on swarms started focusing on heterogeneous UAV swarms comprising multiple UAV types, each designed with distinct roles and capabilities [32]. Compared to homogeneous swarms, heterogeneous UAV swarms offer enhanced scalability and performance, capable of addressing complex and diverse task requirements [9, 19, 45, 47].

An effective software update framework is crucial for swarms, as they often operate in complex field work where direct connectivity is limited or unavailable, such as search-and-rescue or remote surveillance missions, in mission-critical situations [51]. Recalling an entire swarm just to update their control program is not feasible in such settings. This is why prior research has proposed several swarm software update strategies [1, 44]. For example, over-the-air programming (OTAP) [5, 44] allows updates to homogeneous swarms by rebroadcasting the update once it has been received by an individual drone. Software updates in swarms face challenges, such as bandwidth constraints, computational and storage constraints, and communication failures.

Unfortunately, existing software update strategies for swarms all assume (1) homogeneous UAV swarms (i.e., all UAVs have identical roles and update requirements), and (2) software updates comprising typical code and/or data payloads instead of deep learning (DL) model updates. Heterogeneous swarms introduce complications such as multiple software versions and configurations, different hardware and storage capabilities, and incompatibility with certain software. Furthermore, UAVs increasingly adopt DL models, such as convolutional neural networks (CNNs), for perception tasks, which require updates during long-running missions to prevent concept drift [14, 39]. Without reliable and efficient updates, and effective update strategies able to deal with UAVs' limited resources, UAVs risk mission failure from outdated software or DL models.

To address these challenges, we introduce a framework tailored to heterogeneous UAV swarms called **SwarmUpdate**, which has two components: **SwarmSync** and **SwarmModelPatch**. SwarmSync is a hierarchical software update synchronization strategy designed for heterogeneous UAV swarms that categorizes UAVs into sub-swarms to organize the distribution of software patches. SwarmModelPatch is a DL model patching method that focuses on reducing the size of a model update while minimizing degradation

in model accuracy by selectively freezing model layers before tuning. This paper also systematically evaluates SwarmUpdate through comprehensive simulation experiments, comparing its performance against existing baseline methods.

To illustrate the potential impact and importance of our study, consider the following real-world scenario. A surveillance UAV swarm is tasked with patrolling remote mountainous areas far beyond the range of infrastructure support. When this swarm encounters sudden environmental changes (e.g., unexpected snowfall), the DL models on the UAVs' perception systems may become unreliable, severely affecting mission outcomes. The mission cannot afford downtime for updates, yet adaptability is crucial. Instead, the swarm could send a small number of UAVs back carrying any relevant new data, notifying headquarters that an update is required to maintain functionality, or communicate new data back through peer-to-peer communication links between UAVs all the way back to base. In response, headquarters trains the existing model with the new data, then sends an "Updater" UAV back to the swarm, responsible for distributing the updated model to any UAV that requires an update using the SwarmSync synchronization approach. Once a UAV receives the update, it leverages the SwarmModelPatch model patching method to incrementally update its DL model.

The contributions of this paper are summarized as follows:

- **SwarmSync:** A software update synchronization strategy for heterogeneous UAV swarms, leveraging the hierarchical structure of swarms to distribute software patches.
- **SwarmModelPatch:** The first DL model patching method designed for UAV swarms to train DL models to adapt to new data.
- A replication package contains simulation environment setups, scripts, data, and DL models at <https://anonymous.4open.science/r/SwarmUpdate-860D>.

## 2 Related Work

**Software Update Strategies.** Existing research on software update strategies primarily target homogeneous swarms [1, 44]. For instance, Varadharajan et al. [44] propose an update protocol incorporating rebroadcasting and retransmission through requests, while Al Blooshi and Han [1] focus on security threats when updating a swarm. However, these studies do not address the complexities inherent in updating heterogeneous swarms, where different UAVs have distinct roles and diverse update requirements. Additionally, traditional software patching usually replaces the older version, which is unsuitable for resource-constrained UAV swarms with DL models. SwarmUpdate only transmits the differences between model versions. Moreover, by freezing earlier layers, SwarmUpdate minimizes update size and bandwidth usage.

Since egalitarian, homogeneous swarms can be considered as a Peer-to-Peer (P2P) system, swarm researchers have explored various software update strategies for P2P systems [6]. For example, a widely used protocol for P2P system software updates is Gossip-based protocols, Busnel et al. [6] use gossip-based protocols for distributing software updates for wireless sensor networks. However, gossip-based protocols often suffer from redundant messaging, inconsistencies, and lower performance in sparse connections [34].

**Deep Learning Model Updating.** Updating deep learning (DL) models is a critical challenge for updates in UAV swarms, as DL models are usually large and bandwidth-consuming. Researchers have proposed various techniques to mitigate the need for full re-training, such as transfer learning [50] and lifelong learning [29, 30]. The idea of lifelong learning can be traced back to McCloskey and Cohen [23] in 1989, which proposes training a model incrementally on new data. Recently, Olewicki et al. [29] explore cost-efficient lifelong learning techniques to reduce computational overhead compared to retraining. To avoid catastrophic forgetting, a replay buffer is used containing a portion of old training data. Transfer learning leverages pre-trained models for adaptation to new but related tasks [50], lowering the computational resources needed to train. However, these studies primarily focus on improving accuracy or minimizing training resources [12, 17, 38], rather than reducing the size of model updates. To the best of our knowledge, our paper is the first to propose a DL model patching method that aims to minimize model changes and hence to reduce model patch sizes.

While compression techniques like pruning [15] and quantization [12, 37, 52] reduce overall model size and computational cost, they do not specifically focus on minimizing differences between model versions to optimize patch sizes. Pruning removes less important neurons post-training, whereas quantization reduces memory and computation by lowering weight precision (e.g., from 32-bit float to 8-bit int). However, neither explicitly targets reducing the model update size between versions.

**Software Engineering for UAV Systems.** Software engineering researchers have studied different aspects of UAV systems such as dynamic updates, robustness, and vulnerabilities. For example, Nahabedian et al. [28] introduce an automated dynamic controller synthesis approach to facilitate efficient updates of UAV controllers, validated in UAV surveillance systems. To enhance the robustness of UAV systems, Wang et al. [49] present a method for detecting anomalies from UAV logs, while Han et al. [13] implement a learning-guided search framework to detect vulnerabilities in UAV configuration modules. In addition, Jung et al. [18] propose a dedicated debugging system for identifying and resolving configuration-related bugs in UAV swarms.

Regarding system interpretability and safety, Ataiefard et al. [3] propose a hybrid deep neural network method to infer internal states of UAV autopilots by analyzing input-output signal data. To ensure safety in UAV software, Liang et al. [22] empirically evaluate the usage of bounding functions within open-source UAV software frameworks to understand safety-critical concerns. Furthermore, Vierhauser et al. [48] introduce a framework for interlocking Safety Assurance Cases (SACs) to improve operational accountability, where each UAV provides a pluggable SAC demonstrating compliance with infrastructure-defined safety constraints.

None of the above work targets the unique needs and complications a heterogeneous UAV swarm faces to update a DL model in-field, nor attempt to improve the efficiency of software updates in UAV swarms.

## 3 SwarmUpdate

This section presents the problem definition of this study and the two main components of SwarmUpdate : a hierarchical software

**Algorithm 1** SwarmSync's Updater

---

```

1: Signal: Update is available
2: Wait until all leaders get in position      ▶ Until Time-out
3: for each packet in packets do
4:   repeat
5:     Transmit current packet to all leaders
6:   until ACK received from all leaders      ▶ Until Time-out
7: end for
8: if Time-out then
9:   Signal Reappoint Leader
10:  Restart Update Process
11: end if
12: Wait until convergence signals received from all leaders

```

---

update synchronization strategy called SwarmSync and a DL model patching technique called SwarmModelPatch.

**3.1 Problem Definition**

Consider a UAV swarm as a set  $S = \{u_1^{t_1}, u_2^{t_2}, \dots, u_n^{t_n}\}$ , where each UAV  $u_i^{t_i}$  belongs to a type  $t_i$ , runs software with version  $v_i^c$ , and has a given communication range  $r_i$ . A software version  $v^c$  is compatible with UAVs whose type  $t$  satisfies  $t \in c$ , where  $c$  are the compatible types for that software update,  $c = \{t_j | t_j \text{ is compatible with } v^c\}$ . UAVs incompatible with the update (i.e.,  $t \notin c$ ) are excluded from receiving it. Let  $S^c \subseteq S$  represent the subset of UAVs compatible with update  $v^c$ . The goal is to distribute the latest software version  $v_{\text{new}}^c$  to all compatible UAVs in  $S^c$ , ensuring update coverage with minimal latency and overhead.

A subset  $S^m \subseteq S$  of UAVs are equipped with an imaging sensor whose input is fed to a DL CNN model  $\mathcal{M}$  to perform perception tasks. Consider such a model  $\mathcal{M}$  capable of classifying images into classes (labels)  $L = \{l_1, l_2 \dots l_n\}$  with accuracy  $Acc$ . When a new class  $l_{n+1}$  is introduced, the existing model must be updated to a new model version  $\mathcal{M}^*$  capable of classification into  $L^* = \{l_1, l_2 \dots l_n, l_{n+1}\}$  with accuracy  $Acc^*$ . Updating the existing model from  $\mathcal{M}$  to  $\mathcal{M}^*$  generates a model patch  $\mathcal{P}$  with size  $s$ . The goal is to minimize  $s$  with minimal degradation in  $Acc^*$ .

**3.2 SwarmSync**

We propose SwarmSync to ensure efficient and reliable software update synchronization for heterogeneous UAV swarms, basically distributing a software update payload (whether it represents an update to a new model version  $\mathcal{M}^*$ , or more traditional code or data updates). SwarmSync is inspired by the scalable SWARM data replication method [25, 40], where a designated supernode distributes data within server networks to other nodes with the same interest. In SwarmSync, the swarm is divided into smaller sub-swarms based on UAV types, where each sub-swarm consists of only one type of UAVs  $t_i$  and appoints a leader (either randomly or based on predefined rules, such as the lowest UAV identifier ID). Note that there might be multiple sub-swarms for a given UAV type  $t_i$ , to avoid too unbalanced sub-swarms. Updates propagate via a structured and hierarchical mechanism based on three distinct roles: Updater, leader, and follower. The “Updater” UAV is the one physically triggering the update process in close collaboration with

**Algorithm 2** SwarmSync's Leader

---

```

1: Receive signal from Updater
2:  $PatchSize \leftarrow$  from signal, propagate signal to swarm
3: Move toward and surround Updater
4: Signal: leader in position
5: Receive packet from update
6: while #packets <  $PatchSize$  do
7:   Wait to receive the packet      ▶ Until Time-out
8:   if packet received successfully then
9:     Send ACK
10:  end if
11:  if Time-Out then
12:    Abort Update process, signal others to also abort
13:  end if
14: end while
15: Return to respective sub-swarm
16: for each packet in packets do
17:  repeat
18:    Transmit current packet to all followers
19:  until ACK received from all followers  ▶ Until Time-out
20:  if Time-out then
21:    Remove Unresponsive follower from follower list
22:    Continue Next packet
23:  end if
24: end for
25: Signal: Notify Updater that sub-swarm has converged

```

---

**Algorithm 3** SwarmSync's Follower

---

```

1: Receive signal from Updater
2:  $PatchSize \leftarrow$  from signal, propagate signal to neighbours
3: Remain idle until leader returns with update
4: while #packets <  $PatchSize$  do
5:   Wait to receive the packet      ▶ Until Time-out
6:   if Time-out then
7:     Signal Time-out, reappoint leader in the swarm
8:   end if
9:   if packet received successfully then
10:    Send ACK
11:  end if
12: end while
13: Apply Patch
14: Signal: Notify leader completion of update

```

---

the sub-swarms’ “leader” UAVs. The latter then distribute updates to the other UAVs in their sub-swarm, i.e., the “follower” UAVs.

To ensure the guaranteed convergence of the swarm and prevent false positives, SwarmSync uses the Transmission Control Protocol (TCP) [7]. In TCP, every network packet sent requires an acknowledgement (ACK) signal back before the sender moves on to the next packet. In the absence of an ACK signal, the sender keeps on re-sending the current packet. Since each leader tracks its assigned followers, this method guarantees that all drones will receive all packets of the update, which is important as UAVs are often used in mission-critical or safety-critical situations [51]. That said, TCP’s reliance on ACK signals introduces a potential deadlock risk

if a UAV fails to respond. To mitigate potential deadlocks, SwarmSync implements a time-out mechanism: if a drone is unresponsive within a given time frame, it is removed from the leader's follower list; if a leader is unresponsive, the sub-swarm will appoint a new leader in its place. Although a time-out did not occur during the simulation, preventative measures are essential for critical systems.

**Updater.** As presented in Algorithm 1, the Updater signals the swarm that an update is available. This signal contains metadata including version details and update packet size. The Updater then waits for the leaders' signal that indicates they are ready in position. The Updater sequentially sends update packets and waits for ACKs from all leaders before proceeding. After completing transmissions, the Updater remains idle and waits for convergence signals from the leaders. Once the Updater receives convergence signals from all leaders, the swarm is declared to have converged. As the Updater depends on the leaders in two scenarios: (1) waiting for all leaders to get in position before sending an update, and (2) waiting for ACK from all leaders before sending the next packet, a deadlock could happen if a leader becomes unreachable, and thus time-out is introduced for both scenarios. When a time-out happens, a signal is sent out to the swarm, notifying that the leader has been removed from the swarm. The sub-swarm of the missing leader then appoints a new leader and restarts the update process.

**Leader.** As shown in Algorithm 2, when an update signal is received, either from the Updater or from an UAV propagating the update signal, the leader of each sub-swarm moves to the Updater's position and forms a predefined spatial formation (shown in Figure 3) to get in range to receive updates, temporarily leaving its default position.

Once in position, leaders signal their readiness to receive updates. They sequentially receive packets from the Updater and send ACK for each packet. After receiving all packets, leaders return to their respective sub-swarms, distributing the update packets to their followers using the same reliable transfer (TCP) approach. Leaders notify the Updater when all their followers have completed their update. If a leader has no followers, the leader will immediately notify the Updater that the sub-swarm has converged.

To prevent deadlocks, the leader also implements a time-out mechanism, in two scenarios. First, if the leader times out while receiving an update from the Updater, the leader assumes the Updater has a failure and the update cannot be completed. Since only the Updater possesses the update, the process is aborted, and an abort signal is sent by the leader and propagated to other UAVs in the swarm. Second, if the leader times out while waiting for ACKs from the followers, any non-responding follower is removed from the follower list and the update continues for remaining followers.

**Follower.** As the drones have only a limited communication range  $r_i$ , peer-to-peer propagation of data in a swarm is essential in order to reach all UAVs. Shown in Algorithm 3, UAV followers propagate the update signal obtained from their sub-swarm leader to all their neighbors to expedite the signal. While the leaders are getting the update from the Updater, the followers continue their usual activities but remain within the communication range of their leader's last location (before moving to the Updater) until the leader returns to the sub-swarm. The followers will send an ACK for each received packet to their leader. Once all packets are received, the followers will apply the patch and signal their leader

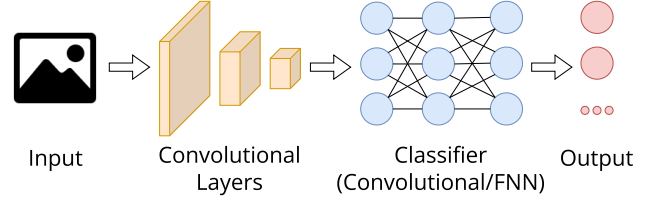


Figure 1: Structure of a typical CNN model.

**Algorithm 4** SwarmModelPatch, referring to Figure 1

---

```

1: Input: Old CNN model  $\mathcal{M}$ , Replay Buffer Size  $\beta$ , Trainable
   Layers  $x$ 
2: Output: Updated model  $\mathcal{M}^*$ , Patch file  $\mathcal{P}$ 
3: Prepare training data:
4:    $\mathcal{D} \leftarrow \text{ReplayBuffer (old data)} \cup \text{New Data}$ 
5: Load model  $\mathcal{M}$ 
6: Reinitialize classifier layer  $C$  in  $\mathcal{M}$ 
7: Freeze all layers except:
8:   - Last  $x$  convolutional layer(s)            $\triangleright$  Adjust accordingly
9:   - Classifier  $C$ 
10:  $\mathcal{M}^* \leftarrow \text{Train } \mathcal{M} \text{ on } \mathcal{D}$             $\triangleright$  Update trainable parameters
11: Initialize empty dictionary  $\mathcal{P}$ 
12: Iterate through parameters of } \mathcal{M}^*:
13: for each parameter  $p$  in  $\mathcal{M}^*$  do
14:   if  $p$  not in  $\mathcal{M}$  then
15:      $\mathcal{P}[p] \leftarrow \mathcal{M}^*[p]$ 
16:   end if
17: end for
18: Save } \mathcal{P} to patch file
19: Return Updated model  $\mathcal{M}^*$ , Patch file  $\mathcal{P}$ 

```

---

about completion. Since the follower depends on the leader to both return and send update packets, a time-out mechanism is in place: the follower assumes that the leader is unreachable and sends a signal to reappoint a new leader for the sub-swarm.

In ideal scenarios, if  $|S^c|$  is the number of UAVs requiring updates, and  $N$  is the maximum number of UAVs updated simultaneously per cycle, where a cycle is the amount of time a drone require to update. For swarm sizes  $|S^c| \leq N^2$ , SwarmSync can complete the update process in two update cycles. For swarms of size  $|S^c| \leq N$ , the Updater will be appointed as the temporary leader and all followers be updated directly by the Updater instead, completing the update process in one update cycle. For  $|S^c| > N^2$ , subsub-swarms can be created recursively to maintain the hierarchical structure of the swarm, where each follower of the sub-swarm can lead their own subsub-swarm and propagate the update. In practice, delays arising from positional adjustments or network conditions may increase this reported optimal complexity.

### 3.3 SwarmModelPatch

Once the payload of a software update is synchronized across all UAVs in a swarm, it needs to be installed. For traditional code/data updates, UAVs would typically “replace” the previous version’s code/data. However, in the case of a deep learning model update,

**Algorithm 5** Apply Model Patch

---

```

1: Input: Base model  $\mathcal{M}$ , Patch file  $\mathcal{P}$ 
2: Output: Updated model  $\mathcal{M}^*$ 
3: for each parameter  $p$  in  $\mathcal{P}$  do
4:   if  $p \in \mathcal{M}$  and  $\text{shape}(\mathcal{M}[p]) = \text{shape}(\mathcal{P}[p])$  then
5:      $\mathcal{M}[p] \leftarrow \mathcal{M}[p] + \mathcal{P}[p]$   $\triangleright$  Update existing parameters
6:   else
7:      $\mathcal{M}[p] \leftarrow \mathcal{P}[p]$   $\triangleright$  Replace new or reshaped parameters
8:   end if
9: end for
10: Return updated model  $\mathcal{M}^*$ 

```

---

a naive “replace” strategy would require an entire new model version to be re-trained/fine-tuned, then sent in the payload of an update. This would not scale for typical swarm settings, where only intermittent peer-to-peer communication is available.

To effectively deal with model updates and reduce the costs of retraining from scratch, we propose SwarmModelPatch, which integrates ideas from lifelong learning [29, 30], transfer learning [20, 27], and incremental learning [36]. As shown in Algorithm 4, SwarmModelPatch maintains a replay buffer holding representative samples from previously seen training data. When an environmental change occurs, the model update incorporates both the replay buffer and new data to prevent catastrophic forgetting [29] and enable rapid adaptation.

A typical CNN model is shown in Figure 1, a CNN model usually consists of some trainable convolutional layers, untrainable pooling layers, and a classifier which can be either a feed-forward neural network (FNN) or convolutional layer(s). The output of the classifier is used to determine the classification results. To efficiently enable a given model to classify data according to newly introduced classes, SwarmModelPatch reinitializes the classifier layer [11, 41], while partially freezing the remaining layers. Once a new model version is trained, SwarmModelPatch generates a model patch representing the differences between the new and old model versions, essentially only containing the changed layers and metadata about these layers (and those they attach to), while ignoring the frozen layers.

Algorithm 5 shows the algorithm of SwarmModelPatch for applying a patch. If a parameter  $p$  in patch  $\mathcal{P}$  has the same shape as the corresponding parameter in model  $\mathcal{M}$ , it is added to  $\mathcal{M}[p]$ . However, if  $p$  does not exist in model  $\mathcal{M}$  or has a different shape,  $\mathcal{M}[p]$  is replaced with  $\mathcal{P}[p]$ .

## 4 Empirical Evaluation Methodology

This section presents the methodology of the empirical evaluation of our proposed software update framework for heterogeneous UAV swarms. We investigate the following research questions (RQs):

**RQ1. What is the update synchronization efficiency of SwarmSync in a heterogeneous UAV swarm?** The efficiency of update synchronization plays a crucial role in determining the optimal strategy for UAV swarms [44]. A slow update process can lead to prolonged idling of the drones, hindering their ability to accomplish their task. Ensuring an efficient update process allows the UAV swarm to continue their

**Algorithm 6** Gossip

---

```

1: Receive signal from Updater
2:  $PatchSize \leftarrow$  from signal, propagate signal to neighbours
3: Receive update packets
4: if #packets <  $PatchSize$  then  $\triangleright$  Packet loss detected
5:   Request retransmission
6:   Receive update packets
7: else
8:   rebroadcasting  $\leftarrow$  true
9: end if
10: if rebroadcasting then
11:   for each packet in update do
12:     Send packet
13:     if request received then
14:       Retransmit packets
15:     else
16:       Signal convergence
17:     end if
18:   end for
19: end if
20: if signal_converged received then
21:   Pass it on
22: end if
23: if all nodes signal_converged then
24:   Converged
25: end if

```

---

ongoing tasks, which directly impacts the swarm’s overall effectiveness and operational strength.

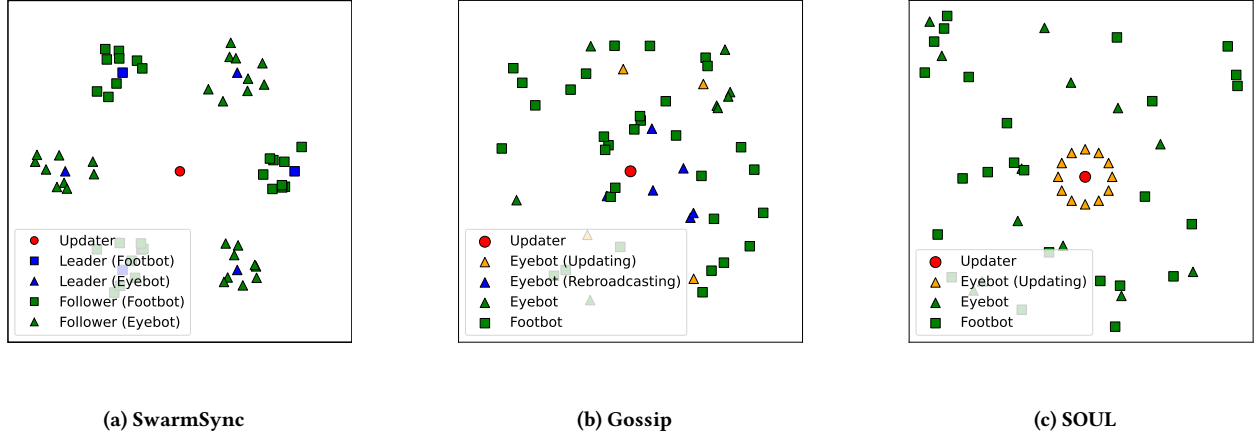
**RQ2. What is the transmission overhead of SwarmSync in a heterogeneous UAV swarm?** Overhead is another important factor in over-the-air software updates [4], directly influencing system performance and efficiency. Overhead consists of communication signals exchanged and the resources consumed for packet transmission. Having excessive overhead can increase computational resource usage, consume more energy, and impact the speed of propagation.

**RQ3. What is the trade-off between efficiency, overhead and accuracy when updating a model with SwarmModelPatch?** Transferring a DL model across a UAV swarm to update to a new model version presents challenges, impacting both the update efficiency and transmission overhead consumption. High update size can cause high transmission costs, which can lead to prolonged update times, excessive overhead, and potential storage or communication bottlenecks, damaging the UAV swarm’s performance.

### 4.1 Baseline Strategies

To empirically evaluate SwarmSync and SwarmModelPatch, we select state-of-the-art baseline techniques. For SwarmSync, we use a gossip-based rebroadcast baseline strategy and an auction-based baseline strategy (both illustrated in Figure 2), while for SwarmModelPatch we select a full-size update patch strategy.

**4.1.1 Gossip-Based Rebroadcast Strategy.** The gossip-based rebroadcasting method follows the principles of epidemic dissemination,



**Figure 2: Swarm UAV formations and roles for SwarmSync and the two baseline software update synchronization strategies.**

modelled by the classic Susceptible-Infected-Recovered (SIR) epidemic model [8]. This model enables information to spread across the swarm with minimal time consumption, even in unreliable UAV communication networks where packet loss and intermittent connectivity are common.

Varadharajan et al. [44] uses such a gossip-based rebroadcasting approach to update a homogeneous swarm. In this strategy, if a receiver fails to compile the update due to incomplete packets, it requests retransmission from the sender, which will resend all packets. Once a drone successfully receives all packets, it transitions into an Updater, actively participating in further dissemination. The new Updater broadcasts the received packets to its neighboring drones, facilitating a decentralized propagation mechanism. However, as the Updaters are unaware of drones around them, they cannot determine when an update is finished. Gossip-based strategies can also generate high transmission overhead due to redundant rebroadcasting, making them inefficient for bandwidth constrained environments such as UAV networks. Additionally, these protocols do not differentiate between heterogeneous UAV types, leading to unnecessary information being propagated to drones that do not require them.

As Gossip has already been implemented to update homogeneous swarms, we adapt it for heterogeneous swarms by introducing a simple modification. Specifically, drones that are not interested in the update will pass on the update message but will not receive or rebroadcast the update (as shown in Algorithm 6). A timeout system similar is introduced: if none of the broadcasting drones receive an update request in 20 control steps of the simulator (see Section 4.2), the swarm is declared as converged.

**4.1.2 Auction-Based Strategy (SOUL).** Auction-based strategies optimize data distribution and overhead by allowing UAVs to bid for content and only transmit necessary data. In this approach, the bidders need to bid for content that the auctioneer will send out after considering all the bids. While auction-based strategies minimize redundant transmissions, they are similar to a client-server model. As a result, when the number of bidders exceeds the maximum

---

**Algorithm 7** SOUL’s Updater distributing update

---

```

1: Signal update is available
2: Signal update first group
3: Wait for signal UAVs are at location
4: Send update
5: while receiving requests do                                ▶ From UAVs
6:   Send requested packets                                     ▶ To UAVs
7: end while
8: Proceed to next group
9: if all groups are done then
10:   Converged
11: end if

```

---

number of connections an auctioneer can handle, update times increase more linearly.

SOUL [46] proposes an auction-based method for data sharing in a swarm. In SOUL, there is an auctioneer that handles the data blob, and it auctions the blob out for the swarm; any UAV in the swarm with available data storage and computation will bid on this blob. Based on the bidding, the auctioneer will decide how to split and distribute the blob to UAVs inside the swarm. Auction-based approaches allow updates to be distributed based on UAVs’ availability, reducing redundant transmissions.

To adapt SOUL for updating a heterogeneous swarm, we set the Updater as the auctioneer and all UAVs as bidders. As shown in Algorithm 7 and Algorithm 8, the Updater broadcasts the entire update, then UAVs that lack certain packets request retransmission via bidding. The Updater selectively rebroadcasts missing packets based on the received bids. UAVs whose positions are outside the Updater’s range move toward the Updater to receive the update. If more than  $N$  UAVs need the update, only  $N$  will be allowed to go to the Updater and update at a given time. In this case, the swarm will form egalitarian groups and be updated group-by-group by the Updater. Similar to SwarmSync, SOUL drones will also propagate the update signal out to their neighbours. Since the Updater is unaware of the status of UAVs, if no request signal is received



**Algorithm 8** SOUL's UAV receiving update

---

```

1: Receive signal from Updater
2:  $PatchSize \leftarrow$  from signal, propagate signal to neighbours
3: Wait for group's turn
4: if updating this group then
5:   Move to update location
6:   Signal at_location
7: end if
8: Receive packets ▷ From Updater
9: while #packets <  $PatchSize$  do
10:   Find missing packets in self storage
11:   Request missing packets
12:   Receive packets
13: end while
14: Signal complete
15: Move away from location

```

---

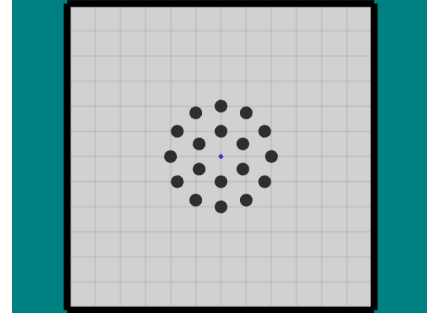
within a certain number of simulator control steps (e.g., 20), the Updater assumes that the group has finished receiving the update.

**4.1.3 Baseline DL Model Patching Method.** No prior research has studied DL model patching for swarms. Since SwarmModelPatch offers adjustable model patch size by adjusting the number of frozen layers, a SwarmModelPatch variant that would not freeze any model layer during fine-tuning or re-training would be equivalent to fine-tuning or re-training the entire model, followed by replacing the previously installed model as a whole. Hence, we consider the SwarmModelPatch variant with zero frozen layers as our baseline, which we refer to as “full-size update”.

## 4.2 Experimental Design

**4.2.1 Simulation Environment.** Our empirical evaluation uses ARGoS [35] as the simulator to model swarm behaviour. ARGoS is a widely-used C++ simulator for robotics designed specifically for swarm simulation. While Gazebo [21] and Webots [24] are also popular robotics simulators, ARGoS emphasizes multi-threaded execution, making it particularly suited for scenarios involving swarms with hundreds of UAVs. Furthermore, ARGoS was also used to evaluate SOUL and Gossip. ARGoS uses “control steps” as a measure of time and efficiency. In the default ARGoS configuration, each control step is 100ms, which we will use in our study to measure the efficiency of the synchronization and patch strategies. Each simulation is ran 10 times and taken the average to ensure the statcal significance.

**4.2.2 Update Configuration.** Similar to earlier research on heterogeneous swarms in ARGoS [9, 10], we simulate a swarm consisting of three types of UAVs, each expecting different types of software updates: the Updater, Footbots, and Eyebots. Specifically, the Updater manages the update and relays it into the swarm, the Eyebots require a DL model update due to sudden weather changes (from sunny to snowy), while the Footbot does not require any update. These updates symbolize a heterogeneous swarm where each type of UAV has its own update requirements. As the ARGoS built-in



**Figure 3: 18 Drones surrounding the Updater.**

“eye-bot”<sup>1</sup> inherently cannot move, it does not represent a field-work use case, we use the “foot-bot” implementation to simulate a heterogeneous swarm composition, as “foot-bot” and “eye-bot” both share the same communication characteristics. We experiment with swarm sizes  $s \in \{20, 100, 200, 500\}$  (excluding the Updater), with an equal ratio of 1 : 1 between Footbots and Eyebots.

**4.2.3 SqueezeNet Model.** We choose SqueezeNet [16] as the DL model for the Eyebots due to its lightweight architecture that modern drone hardware can handle [26]. For our evaluation, we select a 5-class weather classification task [2] to train the model. The dataset contains five classes: Sunny, Cloudy, Foggy, Rainy, and Snowy. To simulate environmental concept drift, we exclude the Snowy class from the original training data and consider Snowy as the new class in our simulation. After training SqueezeNet on the remaining 4 classes, we apply SwarmModelPatch to generate a model patch for a new model version that includes Snowy, with the replay buffer keeping 40% of the old training data and combined with Snowy to obtain the final training data. As the dataset did not provide a balanced number of data per class nor was there inherent train/test seperation, we performed data augmentation to upscale all other classes to the class with the most data (Sunny with 6274 images), then we split the data into train/test/val (0.7:0.15:0.15). Each model was trained 3 times and the best model is kept to study. To clearly analyze the impact of SwarmModelPatch on update size patterns and times, we do not apply quantization and other compression techniques that could further reduce patch sizes.

In SqueezeNet, convolutional layers are represented as “fire modules”, where each of the eight fire modules consists of multiple layers. For this study, we treat each fire module as a convolutional “layer” that could potentially be frozen. SqueezeNet also uses a convolutional layer as the classifier instead of a feed-forward neural network (FNN); this layer is unfrozen by default.

**4.2.4 Communication Range.** As there are physical limitations for the physical formation of drones (e.g., two drones cannot move into the same location), and ARGoS used a default communication length of  $r = 3m$ , we standardize the maximum number of drones simultaneously updating around the Updater to  $N = 18$ . This number is derived using a hexagonal distribution of drones and ARGoS’ communication range, as shown in Figure 3.

<sup>1</sup>[https://pinciroli.net/api/dir\\_8bd2882cf9d14055c3b0116edcf0e192.php](https://pinciroli.net/api/dir_8bd2882cf9d14055c3b0116edcf0e192.php)

**4.2.5 Communication Failure.** We implement packet-loss scenarios as an all-or-nothing model using the ARGoS default configuration, meaning that if a packet is determined to be lost, the receiver will receive nothing. We use the same packet failure rates  $f \in \{0, 0.25, 0.50, 0.75\}$  as a previous study on homogeneous swarms [44].

**4.2.6 Transmission Speed.** While ARGoS' default data transmission speed is 100 bytes/sec, this is an artificial constraint that does not reflect real-world UAV communication speeds. We instead set a transmission speed of 1 Mbps based on the Bitcraze Crazyradio 2.0 module,<sup>2</sup> which is widely used in swarm robotics research. Since the default ARGoS configuration defines one control step as 100 ms, this means there are 10 control steps (i.e., 10 packets) per second. To achieve a 1 Mbps transmission speed (equivalent to 125 kB per second), we set the packet size to 12.5 kB, ensuring that 10 packets per second result in the desired throughput.

**4.2.7 Swarm Behaviour.** In our simulation, the Updater will be deployed and remain stationary at (0, 0), while all Footbots and Eyebots are distributed randomly following a diffusion pattern for surveillance [33]. By default, the topology of the swarm is the cluster topology [34] and the maximum speed for the drones is  $V = 1m/s$ , aligning with previous research [43].

## 5 Results

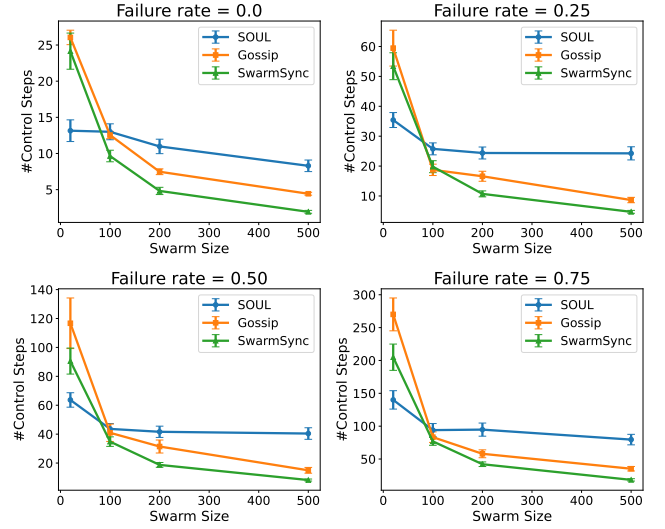
### 5.1 RQ1: What is the update synchronization efficiency of SwarmSync in a heterogeneous UAV swarm?

**Setup.** To compare the efficiency of SwarmSync's with the two baseline update synchronization strategies, we use the baseline DL model update method ("full-size update") instead of Swarm-ModelPatch (which is evaluated in RQ3). As such, model updates correspond to 240 packets each. The efficiency of update synchronization is measured as the number of control steps from the start of the update synchronization until the swarm declares convergence.

**5.1.1 Findings. SwarmSync achieves the fastest convergence time across all strategies, outperforming SOUL by 78.3% and Gossip by 47.7% on average for the largest swarm size of 500.** Figure 4 presents the average time taken across all drones to converge for varying swarm sizes. Compared to SOUL and Gossip, SwarmSync demonstrates efficient update propagation. At the largest swarm size 500, across all failure rates, SwarmSync completes the update on average 78.3% faster compared to SOUL, and on average 47.7% faster compared to Gossip. These results reinforce that SwarmSync is an efficient strategy for updating UAV swarms. However, SwarmSync shows minor inefficiencies for small swarms (e.g., 20), which is likely due to a limitation in ARGoS where packets and their ACKs cannot be handled in the same control step, leading to wasted control steps that would not occur in real-life settings, this is discussed more in Section 7.

**SwarmSync scales efficiently with increasing swarm sizes.**

As the swarm size increases from 100 to 500 for failure rate 0.0, the average number of control steps decreased from 9.7 to 1.9 per drone for SwarmSync, showing a reduction by a factor of 5.1. This means that SwarmSync decreases the update synchronization



**Figure 4: The average time taken per drone for each synchronization strategy to converge the swarm.**

time per drone approximately inversely proportional to the size increase; when the swarm size increased by a factor 5 times, the time per drone decreases by a similar factor of 5.1, implying more predictable and efficient scaling. SOUL exhibits higher convergence times across all failure rates, showing a time per drone trend almost parallel to the x-axis after it almost stabilizes after swarm size 100. While Gossip performs better than SOUL, it still shows a higher increase in convergence time as swarm sizes increase.

### 5.2 RQ2: What is the transmission overhead of SwarmSync in a heterogeneous UAV swarm?

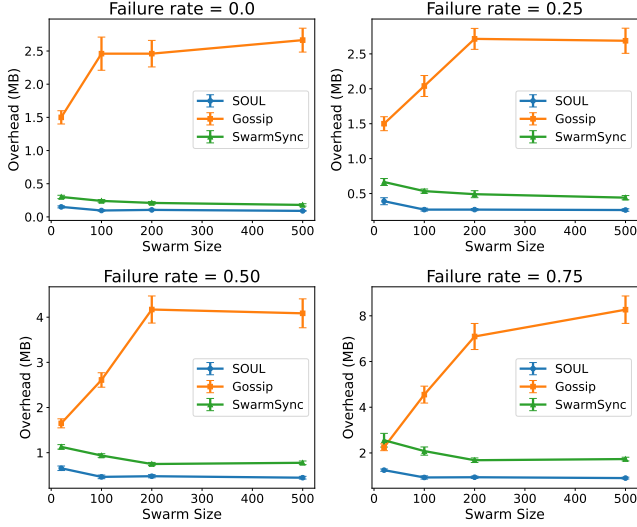
**5.2.1 Setup.** Using the same setup as RQ1, we evaluate the transmission overhead of the updates, which is accounted for when a drone sends out a packet (12.5 kb), and when a drone sends out a communication signal (1 byte to 10 bytes), the average size of communication signal is 5 bytes. We measure the total overhead from the start of the update synchronization until the swarm converges in Megabytes. The update payloads remain the same as for RQ1.

**5.2.2 Findings. SwarmSync generates an average of 87.7% more overhead than SOUL but only produces 19.7% overhead compared to Gossip.** Figure 5 shows, for both SwarmSync and SOUL, that transmission overhead decreases as swarm size increases. While SwarmSync produces an average of 87.7% more overhead than SOUL across all failure rates and swarm sizes, the overhead for both SOUL and SwarmSync presents a decreasing trend as swarm size increases. In contrast, Gossip creates up to 8.3 MB of transmission overhead at a failure rate of 0.75, showing a 72.8% increase in overhead compared to SwarmSync at a swarm size of 20.

**Higher failure rates increase overhead across all three strategies, at swarm size 20, SOUL and SwarmSync show similar overhead growth (828.4% and 850.5%) as failure rate rises from 0.0 to 0.75.** As shown in Figure 5, the overhead for all strategies grows with increased failure rates across different swarm size

<sup>2</sup><https://www.bitcraze.io/products/crazyradio-2-0>





**Figure 5: The average overhead per drone for each synchronization strategy until convergence of the swarm.**

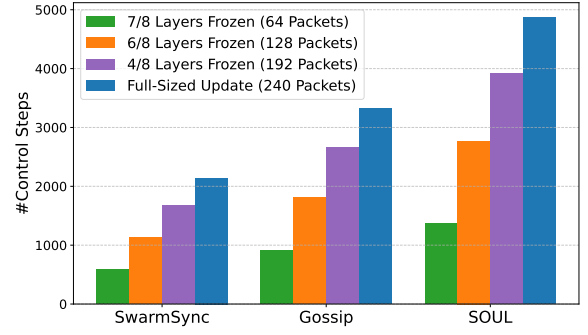
settings. Notably, the relative overhead ratios among the strategies remain similar as the failure rate grows. Both SwarmSync and SOUL almost double in terms of overhead with each increase of the failure rate. Interestingly, while the overhead of both SwarmSync and SOUL grew by more than 100% from failure rate 0 to 0.25, the overhead for Gossip remained relatively stable.

However, both SOUL and SwarmSync maintain a relatively stable overhead increase (around 15 times) when swarm size grows from 20 to 500 across all failure rates. The overhead for Gossip does not remain stable, with the same increase in swarm size, the increase in overhead goes from around 40 times for failure rate 0 and 0.25, to 62 times at failure rate 0.5, and 91 times at failure rate 0.75. This is likely due to the formation of Gossip, where lower sizes have multiple Updaters broadcasting to the same receivers, making the receiver get the update with minimal packet loss. With the size of the swarm increasing, the length between the Updater and the drones on the edge of the swarm increases, creating the need to get the update relayed through the entire swarm, growing in overhead.

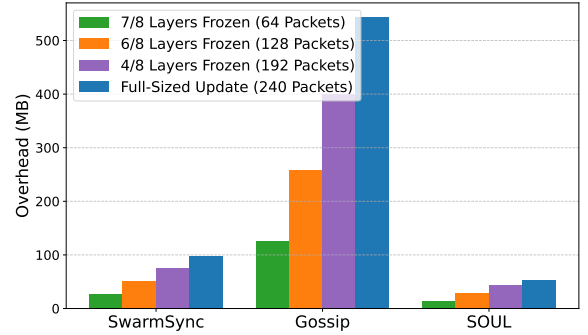
### 5.3 RQ3: What is the trade-off between efficiency, overhead and accuracy when updating a model with SwarmModelPatch?

**5.3.1 Setup.** To evaluate SwarmModelPatch's ability to reduce model patch size while maintaining model accuracy, we progressively freeze the model layers up to the last layer ("fire module" in SqueezeNet), performing model updates for each number of frozen layers and comparing the trade-off between accuracy and update size with the "full-size update" baseline that was used in RQ1 and RQ2, each model is trained with 10 epochs. In SqueezeNet, layers closer to the classifier contain more weights, meaning that freezing those layers reduces the update size of the patch more than the other (more shallow) layers.

In particular, we select three configurations for our experiment: freezing 4, 6, and 7 fire modules. These configurations produce



**Figure 6: The average number of control steps for swarm convergence across varying patch sizes.**



**Figure 7: The average overhead for swarm convergence across varying patch sizes.**

balanced reductions in patch size, with the configuration freezing 4 fire modules yielding a patch size approximately 300% larger than the one obtained by freezing 7 modules, and freezing 6 fire modules resulting in a patch size approximately 200% larger than the 7-module case. For example, the full-size update used in RQ1 and RQ2 requires 240 packets for one model version update. Freezing 4 fire modules reduces the model patch size to 192 packets, whereas freezing 6 and 7 fire modules further reduces the update sizes to 128 and 64 packets, respectively. We evaluate the updated models based on two metrics: overall classification accuracy (across all classes, including the new class data) and accuracy specifically on the newly introduced class data.

In addition, we evaluate the efficiency and overhead of different patch sizes in a swarm of 200 UAVs with a failure rate of  $f = 0.25$ . We select swarm size = 200 to get 100 drones to be updated aligning with previous work [44]. We select  $f = 0.25$  because it is a common failure rate for sophisticated UAV systems [31]. We measure the efficiency and overhead with the same measurements as RQ1 and RQ2. Similar to RQ1 and RQ2, we study all three strategies and the impact of patch sizes on them.

**5.3.2 Findings. SwarmModelPatch decreases patch size by up to 73.3%, at the expense of a drop of 5.1% in overall accuracy (from 72.7% down to 67.6%).** Table 1 presents the trade-off in accuracy when applying different numbers of frozen layers to SwarmModelPatch. As more layers get frozen, the overall accuracy

**Table 1: Results of SwarmModelPatch for different numbers of frozen layers.**

Model	Acc (%)	New Class Acc (%)	Size (MB)	Patch Size (MB)
0/8 Frozen (Baseline)	72.7	72.3	2.9	2.9
4/8 Frozen	71.1	68.1	2.9	2.4
6/8 Frozen	68.6	67.4	2.9	1.5
7/8 Frozen	67.6	62.8	2.9	<b>0.8</b>

lowers, from 72.7% down to 67.6% at 7/8 layers frozen, while the size of the patch decreases gradually from 2.9 MB down to 0.8 MB at 7/8 layers frozen. The new class (Snowy) accuracy decreased by 9.5% from 72.3% down to 62.8% at 7/8 frozen layers, showing a heavier impact than the overall accuracy.

**SwarmModelPatch with frozen layers improves up to an average 72.3% convergence speed at 7/8 layers frozen compared to full-size updates.** Figure 6 compares the number of control steps required for different update sizes under a failure rate of 0.25 with a swarm size of 200 (100 Eyebots and 100 Footbots). The results indicate that leveraging SwarmModelPatch with frozen layers improves convergence speed by an average of up to 72.3%, with an error margin of 1%, compared to updating the whole model.

**SwarmModelPatch with frozen layers has up to an average 74.3% lower transmission overhead at 7/8 layers frozen across all strategies.** Figure 7 presents the average transmission overhead (in Megabytes) evaluated under the same parameters as Figure 6. Similar to Figure 6, having a reduced size patch lowers the overhead transmitted during the update. The overhead for both SOUL and SwarmSync was reduced by up to 73.0% (72.9% and 73.0%) for the 7/8 frozen layer update, while the overhead for Gossip was reduced by up to 76.7%, which is likely due to UAVs in Gossip receiving packets from multiple senders at the same time.

## 6 Discussion

**Selecting appropriate UAVs as leaders to improve overall update efficiency.** In our simulation, UAV leaders are randomly assigned inside the sub-swarm since all UAVs are identical. In practice, having leaders equipped with more powerful physical capabilities could enhance the performance of the swarm [45]. Practitioners should select leaders based on key performance indicators such as communication bandwidth, computational power, data storage capacity, and battery life. Prioritizing UAVs with enhanced resources as leaders could reduce synchronization time and increase update efficiency in real-world scenarios.

**Future work should investigate security threats when updating heterogeneous swarms.** Our study leverages a structured, hierarchical update mechanism designed for heterogeneous swarm scenarios. However, the security threats during updates in these heterogeneous settings remain unexplored. Although prior studies, such as Al Blooshi and Han [1], have investigated security threats of software updates for homogeneous UAV swarms, no studies address security concerns specific to heterogeneous swarms. Software update for heterogeneous swarms could introduce unique attack vectors, where adversaries may target role-specific vulnerabilities.

Hence, future research should focus on: (1) understanding the vulnerabilities specific to heterogeneous swarms, and (2) proposing approaches to enhance security (e.g., role-based authentication).

**Future work should explore SwarmModelPatch in other resource-constraint scenarios.** The proposed model patching algorithm, SwarmModelPatch, is specifically designed to tackle resource constraints inherent in UAV swarms scenarios, where limited bandwidth, computational capability, and energy constraints are major bottlenecks. SwarmModelPatch could be applied in other resource-constrained domains, for example, the Internet of Things (IoT) or edge computing platforms. Researchers should explore adapting and evaluating SwarmModelPatch in these scenarios.

## 7 Threat to Validity

**Internal Validity.** ARGOS simulates time in discrete control steps, but its implementation of message transmission and reception introduces a limitation: the receiver can only read signals from the last control step. When a packet is sent at time  $t$ , the receiver can only receive the packet at the next control step  $t + 1$ . Hence, even if the receiver would immediately send an ACK back to the sender, the sender would receive the ACK at time  $t + 2$ . In an optimized control system, at failure rate  $f = 0$ , when a packet is sent at time  $t$ , the ACK packet will also be sent back at time  $t$ , and the sender could move on to the next packet at time  $t + 1$ , yet ARGOS currently does not allow this, making SwarmSync take longer in the simulation. This effect does not affect the other strategies, as other strategies are not reliant on a response before sending the next packet.

In our simulation, we used Footbots and Eyebots to model a heterogeneous UAV swarm, as this is a common setup in ARGOS for such studies. However, this choice may not fully reflect real-world heterogeneous swarm configurations. We maintained an equal ratio of Footbots and Eyebots, though practical applications may vary. Additionally, in ARGOS, we represented the drones' communication range as a circular area, whereas in real-world scenarios, this range is typically more spherical.

**External Validity.** In the simulation, it is assumed that all drones are connected to the swarm and that each drone has at least one relay path to receive information. While this is a common assumption for simulation [34], depending on the situation, this assumption might not hold true for real-world use cases.

In addition, the packet loss is modelled as an all-or-nothing drop. In real-world settings, this is likely not the case. Preventative measures such as hashing should be used to confirm the integrity of the packet. If hashing fails for a packet, then the packet is declared as corrupted, and should be requested again as if it was missing. We also acknowledge that in real-world implementations, a static 1Mb/s transfer speed is not realistic, adaptive bitrate selection may be needed based on interference and range limitations.

We evaluated SwarmModelPatch using SqueezeNet on the 5-class weather classification dataset [2]. Note that the trend of model performance observed with different numbers of frozen layers may differ when applying SwarmModelPatch to other DL models or datasets.

## 8 Conclusion

This study presents SwarmUpdate, the first framework for software updates in heterogeneous UAV swarms, comprising the SwarmSync update synchronization approach and SwarmModelPatch DL model patching approach. We empirically evaluated SwarmSync by comparing it against two baseline update synchronization strategies (SOUL and Gossip) and one baseline model update approach (full-size update) in heterogeneous swarm simulations. Our results show that SwarmSync is the most efficient and scalable out of the three synchronization strategies, achieving up to 78.3% faster convergence times than SOUL and 47.7% faster convergence than Gossip, while maintaining a reasonable overhead of 19.7% compared to Gossip. In addition, our findings confirm the viability of incremental model updates using SwarmModelPatch, showing how it can reduce update time and overhead by up to 72.3% and 74.3%, respectively, without substantially compromising model accuracy.

This research lays a foundation for future studies on software update frameworks for heterogeneous swarm systems, enhancing their reliability and effectiveness. As UAV technology continues to advance, ensuring robust, efficient, and scalable updates will be crucial for mission-critical and large-scale UAV deployments (e.g., humanitarian and surveillance missions).

## 9 Acknowledgements

We extend our gratitude to Vivek Shankar Varadharajan for his invaluable support in providing the initial guidelines and source codes for [44, 46] that contributed to this work.

## References

- [1] Shamma Al Blooshi and Kyusuk Han. 2022. A study on employing UPTANE for secure software update OTA in drone environments. In *2022 IEEE international conference on omni-layer intelligent systems (COINS)*. IEEE, 1–6.
- [2] Ammar Alfaifi. [n.d.]. 5-class weather status image classification. <https://www.kaggle.com/datasets/ammaraifaifi/5class-weather-status-image-classification>
- [3] Foozhan Ataiefard, Mohammad Jafar Mashhadi, Hadi Hemmati, and Neil Walkinshaw. 2022. Deep State Inference: Toward Behavioral Model Inference of Black-Box Software Systems. *IEEE Transactions on Software Engineering* 48, 12 (2022), 4857–4872. doi:10.1109/TSE.2021.3128820
- [4] Jan Bauwens, Peter Ruckebusch, Spiros Giannoulis, Ingrid Moerman, and Eli De Poorter. 2020. Over-the-Air Software Updates in the Internet of Things: An Overview of Key Principles. *IEEE Communications Magazine* 58, 2 (2020), 35–41. doi:10.1109/MCOM.001.1900125
- [5] Stephen Brown and Cormac J Sreenan. 2013. Software updating in wireless sensor networks: A survey and lacunae. *Journal of Sensor and Actuator Networks* 2, 4 (2013), 717–760.
- [6] Yann Busnel, Marin Bertier, Eric Fleury, and Anne-Marie Kermarrec. 2007. Gcp: Gossip-based code propagation for large-scale mobile wireless sensor networks. *arXiv preprint arXiv:0707.3717* (2007).
- [7] Vinton Cerf and Robert Kahn. 1974. A protocol for packet network intercommunication. *IEEE Transactions on communications* 22, 5 (1974), 637–648.
- [8] Daryl J Daley and David G Kendall. 1964. Epidemics and rumours. *Nature* 204, 4963 (1964), 1118–1118.
- [9] Frederick Ducatelle, Gianni A Di Caro, and Luca M Gambardella. 2010. Cooperative self-organization in a heterogeneous swarm robotic system. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. 87–94.
- [10] Frederick Ducatelle, Gianni A Di Caro, Carlo Pinciroli, and Luca M Gambardella. 2011. Self-organized cooperation between robotic swarms. *Swarm Intelligence* 5 (2011), 73–96.
- [11] Yaël Frégier and Jean-Baptiste Gouray. 2021. Mind2Mind: transfer learning for GANs. In *Geometric Science of Information: 5th International Conference, GSI 2021, Paris, France, July 21–23, 2021, Proceedings 5*. Springer, 851–859.
- [12] Karan Goel, Albert Gu, Yixuan Li, and Christopher Ré. 2020. Model patching: Closing the subgroup performance gap with data augmentation. *arXiv preprint arXiv:2008.06775* (2020).
- [13] Ruidong Han, Chao Yang, Siqi Ma, JiangFeng Ma, Cong Sun, Juanru Li, and Elisa Bertino. 2022. Control parameters considered harmful: detecting range specification bugs in drone configuration modules via learning-guided search. In *Proceedings of the 44th International Conference on Software Engineering (Pittsburgh, Pennsylvania) (ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 462–473. doi:10.1145/3510003.3510084
- [14] Manzoor Ahmed Hashmani, Syed Muslim Jameel, Hitham Alhussain, Mobashar Rehman, and Arif Budiman. 2019. Accuracy performance degradation in image classification models due to concept drift. *International Journal of Advanced Computer Science and Applications* 10, 5 (2019), 422–425.
- [15] Yang He and Lingao Xiao. 2023. Structured pruning for deep convolutional neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence* 46, 5 (2023), 2900–2919.
- [16] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size. *arXiv preprint arXiv:1602.07360* (2016).
- [17] Gabriel Ilharco, Mitchell Wortsman, Samir Yitzhak Gadre, Shuran Song, Hananeh Hajishirzi, Simon Kornblith, Ali Farhadi, and Ludwig Schmidt. 2022. Patching open-vocabulary models by interpolating weights. *Advances in Neural Information Processing Systems* 35 (2022), 29262–29277.
- [18] Chijung Jung, Ali Ahad, Jinho Jung, Sebastian Elbaum, and Yonghwi Kwon. 2021. Swarmbug: debugging configuration bugs in swarm robotics. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (Athens, Greece) (ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 868–880. doi:10.1145/3468264.3468601
- [19] Gal A Kaminka and Yinon Douchan. 2025. Heterogeneous foraging swarms can be better. *Frontiers in Robotics and AI* 11 (2025), 1426282.
- [20] Andrej Karpathy, Justin Johnson, and Fei-Fei Li. [n.d.]. Transfer Learning and Fine-tuning Convolutional Neural Networks. <https://cs231n.github.io/transfer-learning/>.
- [21] Nathan Koenig and Andrew Howard. 2004. Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sendai, Japan, 2149–2154.
- [22] Xiaozhou Liang, John Henry Burns, Joseph Sanchez, Karthik Dantu, Lukasz Ziarek, and Yu David Liu. 2021. Understanding Bounding Functions in Safety-Critical UAV Software. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 1311–1322. doi:10.1109/ICSE43902.2021.00119
- [23] M McCloskey and NJ Cohen. 1989. Catastrophic interference in connectionist networks: the sequential learning problem. *Psychol Learn Motiv* 24: 109–165.
- [24] O. Michel. 2004. Webots: Professional Mobile Robot Simulation. *Journal of Advanced Robotics Systems* 1, 1 (2004), 39–42. <http://www.ars-journal.com/International-Journal-of-Advanced-Robotic-Systems/Volume-1/39-42.pdf>
- [25] Behnaz Mohammadi and Nima Jafari Navimipour. 2019. Data replication mechanisms in the peer-to-peer networks. *International Journal of Communication Systems* 32, 14 (2019), e3996.
- [26] Minu MS, Subashka Ramesh SS, et al. 2022. Optimal Squeeze Net with Deep Neural Network-Based Aerial Image Classification Model in Unmanned Aerial Vehicles. *Traitement du Signal* 39, 1 (2022).
- [27] Derrick Mwitii. [n.d.]. Transfer Learning Guide: A Practical Tutorial With Examples for Images and Text in Keras. <https://neptune.ai/blog/transfer-learning-guide-examples-for-images-and-text-in-keras>.
- [28] Leandro Nahabedian, Victor Braberman, Nicolás D’Ippolito, Shinichi Honiden, Jeff Kramer, Kenji Tei, and Sebastián Uchitel. 2020. Dynamic Update of Discrete Event Controllers. *IEEE Transactions on Software Engineering* 46, 11 (2020), 1220–1240. doi:10.1109/TSE.2018.2876843
- [29] Doriane Olewicki, Sarra Habchi, Mathieu Nayrolles, Mojtaba Faramarzi, Sarath Chandar, and Bram Adams. 2024. On the Costs and Benefits of Adopting Lifelong Learning for Software Analytics-Empirical Study on Brown Build and Risk Prediction. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*. 275–286.
- [30] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. 2019. Continual lifelong learning with neural networks: A review. *Neural networks* 113 (2019), 54–71.
- [31] Enrico Petritoli, Fabio Leccese, and Lorenzo Ciani. 2018. Reliability degradation, preventive and corrective maintenance of UAV systems. In *2018 5th IEEE International Workshop on Metrology for AeroSpace (MetroAeroSpace)*. IEEE, 430–434.
- [32] Carlo Pinciroli and Giovanni Beltrame. 2016. Buzz: An extensible programming language for heterogeneous swarm robotics. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 3794–3800.
- [33] Carlo Pinciroli and contributors. 2015. ARGOS3 Examples. <https://github.com/ilpincy/argos3-examples>
- [34] Carlo Pinciroli, Adam Lee-Brown, and Giovanni Beltrame. 2016. A tuple space for data sharing in robot swarms. (2016).
- [35] Carlo Pinciroli, Vito Trianni, Rehan O’Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Mauro Birattari, Luca Maria Gambardella, and Marco Dorigo. 2012. ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm Intell.* 6, 4 (Dec. 2012), 271–295.

- [36] Robi Polikar, Lalita Upda, Satish S Upda, and Vasant Honavar. 2001. Learn++: An incremental learning algorithm for supervised neural networks. *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)* 31, 4 (2001), 497–508.
- [37] Babak Rokh, Ali Azarpeyvand, and Alireza Khanteymoori. 2023. A comprehensive survey on model quantization for deep neural networks in image classification. *ACM Transactions on Intelligent Systems and Technology* 14, 6 (2023), 1–50.
- [38] Sara Romiti, Christopher Inskip, Viktoriia Sharmanska, and Novi Quadrianto. 2022. Realpatch: A statistical matching framework for model patching with real samples. In *European Conference on Computer Vision*. Springer, 146–162.
- [39] Danilo Sato, Arif Wider, and Christoph Windheuser. 2019. Continuous delivery for machine learning: Automating the end-to-end lifecycle of machine learning applications. *Martin Fowler* (2019).
- [40] Haiying Shen, Guoxin Liu, and Harrison Chandler. 2015. Swarm Intelligence Based File Replication and Consistency Maintenance in Structured P2P File Sharing Systems. *IEEE Trans. Comput.* 64, 10 (2015), 2953–2967. doi:10.1109/TC.2015.2389845
- [41] Zhiqiang Shen, Zechun Liu, Jie Qin, Marios Savvides, and Kwang-Ting Cheng. 2021. Partial is better than all: Revisiting fine-tuning strategy for few-shot learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 9594–9602.
- [42] David St-Onge, Vivek Shankar Varadharajan, Ivan Švogar, and Giovanni Beltrame. 2020. From design to deployment: decentralized coordination of heterogeneous robotic teams. *Frontiers in Robotics and AI* 7 (2020), 51.
- [43] Daniel H Stolfi and Grégoire Danoy. 2024. An ARGoS plug-in for the Crazyflyie drone. *arXiv preprint arXiv:2401.16948* (2024).
- [44] Vivek Shankar Varadharajan, David St Onge, Christian Guß, and Giovanni Beltrame. 2018. Over-the-Air Updates for Robotic Swarms. *IEEE Software* 35, 2 (2018), 44–50. doi:10.1109/MS.2018.111095718
- [45] Vivek Shankar Varadharajan, Karthik Soma, Sepand Dyanatkar, Pierre-Yves Lajoie, and Giovanni Beltrame. 2024. Hierarchies define the scalability of robot swarms. *arXiv preprint arXiv:2405.02417* (2024).
- [46] Vivek Shankar Varadharajan, David St-Onge, Bram Adams, and Giovanni Beltrame. 2020. Soul: Data sharing for robot swarms. *Autonomous Robots* 44, 3 (2020), 377–394.
- [47] Vivek Shankar Varadharajan, David St-Onge, Bram Adams, and Giovanni Beltrame. 2020. Swarm relays: Distributed self-healing ground-and-air connectivity chains. *IEEE Robotics and Automation Letters* 5, 4 (2020), 5347–5354.
- [48] Michael Vierhauser, Sean Bayley, Jane Wyngaard, Wandu Xiong, Jinghui Cheng, Joshua Huseman, Robyn Lutz, and Jane Cleland-Huang. 2021. Interlocking Safety Cases for Unmanned Autonomous Systems in Shared Airspaces. *IEEE Transactions on Software Engineering* 47, 5 (2021), 899–918. doi:10.1109/TSE.2019.2907595
- [49] Dinghua Wang, Shuqing Li, Guanping Xiao, Yepang Liu, Yulei Sui, Pinjia He, and Michael R. Lyu. 2024. An Exploratory Investigation of Log Anomalies in Unmanned Aerial Vehicles. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) (ICSE '24). Association for Computing Machinery, New York, NY, USA, Article 210, 13 pages. doi:10.1145/3597503.3639186
- [50] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. 2016. A survey of transfer learning. *Journal of Big data* 3 (2016), 1–40.
- [51] Liudong Xing and Barry W Johnson. 2022. Reliability theory and practice for unmanned aerial vehicles. *IEEE Internet of Things Journal* 10, 4 (2022), 3548–3566.
- [52] Yiren Zhou, Seyed-Mohsen Moosavi-Dezfooli, Ngai-Man Cheung, and Pascal Frossard. 2018. Adaptive quantization for deep neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.