Decentralizing EXAMM: A Peer-to-Peer Architecture for Fault-Tolerant Evolutionary Model Training

Landon Spitzer
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY
lbs9440@rit.edu

Ashrith Mudundi
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY
avm1998@rit.edu

Diana Velychko
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY
dv6943@rit.edu

***Abstract*—EXAMM is a successful neuroevolutionary framework for developing recurrent neural networks, but its manager-worker architecture centralized scheduling, selection, crossover, and global state in a single node, turning that node into both the system's throughput bottleneck and its greatest failure risk. This centralization constrained scalability, left distributed compute underused, and made long-running training sessions vulnerable to collapse when failures occurred. This work therefore restructured EXAMM as a peer-to-peer architecture to remove the manager as a fixed point of coordination and failure. Instead, each peer maintained its own evolutionary islands, shared strong genomes directly with other peers, and used distributed ownership, backup, and recovery mechanisms to preserve progress without centralized control.**

In 600-second experiments on the NGAFID C172 time-series dataset using four peers, the fault-free peer-to-peer system processed genomes about 2.5 times faster and reduced final prediction error by 23%, improving the final MSE from 0.00215 to 0.00164 as compared to the centralized EXAMM baseline. With snapshot-based recovery enabled, the system still outperformed centralized EXAMM after losing 1 of 4 peers, finishing with a 10% lower final error, and it remained competitive even after losing 3 of 4 peers for a 60-second downtime window, recovering to a final MSE of 0.00206 compared with 0.00215 for the centralized baseline without failures. These results showed that decentralization improved both fault tolerance, as well as the speed and effectiveness of evolutionary search.

***Index Terms*—neuroevolution, peer-to-peer, fault tolerance, distributed evolutionary computation, island model, EXAMM, MPI, token ring**

I. Introduction

Neuroevolution presents an automated approach to choosing a suitable architecture for any dataset based on evolutionary algorithms (EA). EXAMM is a distributed neuro-evolution framework that relies on a centralized manager–worker architecture, where the manager controls model scheduling, selection, and crossover [1]. This star topology collects the control of genome generation, scheduling, pairing (e.g., crossover selection), and system state at the manager, and each worker evolves its island population through model training. Centralizing a system creates a "weak" point at which failure of one element can cause the failure of the whole system and potential loss of produced models. While conceptually simple and widely adopted [2], this centralized design introduces fundamental scalability limitations. This design also limits the scalability of the distributed system, increasing processing pressure and creating a performance bottleneck. Concentrating the distribution of genomes in one manager node leads to idle computational resources and constrains the ability of neuroevolutionary systems to scale with modern distributed hardware.

Therefore, we are proposing to restructure a centralized manager-worker system into a decentralized, peer-to-peer connected system of workers with distributed memory. The aim of this project is to create a fault-proof decentralized system protected from the loss of information and to introduce a topology-aware crossover operation.

A. Importance of Challenge

In the manager-worker system, the manager node holds the whole population and manages selection and evolutionary computation, which entails disproportionate usage of operational memory, storage, and computational power, creating a heterogeneous workload. This challenge is significant because it represents a ceiling on computational scalability and fault tolerance.

- **The Single Point of Failure:** If the Manager node crashes, the entire training session, which can take up to weeks of CPU time, can be lost or corrupted if the database state was not perfectly synchronized or preserved. If the Manager or any worker crashes, the

entire session ends within the EXAMM framework. After a crash, the neuroevolutionary process is killed and has to be restarted. It would be technically possible to load the best genome; however, this mechanism is not implemented in the original code, moreover, it would hinder further progress and limit the search of the neural architecture to one direction determined by a single genome, greatly increasing the chances of getting stuck at the local minimum.

- **The Communication Bottleneck:** In Manager-Worker systems, all data must flow through the Manager's network interface. As a result, as the number of workers scales, the Manager's bandwidth becomes the constraint.
  - **Early stage frequency bottleneck:** In the early stage of neuroevolution, Workers finish fitness evaluations quickly (due to the low complexity of the models), producing a high arrival rate of result submissions and genome requests that overwhelms the Manager's ingress capacity. From a queuing theory perspective, even if each individual message is small, the aggregate inter-arrival time between requests collapses. This pushes the manager toward a hot spot, creating a fan-in bottleneck. The system behaves less like a distributed framework and more like a star topology under thundering herd conditions, where the Manager's throughput becomes the primary constraint on generational throughput.
  - **Late stage throughput bottleneck:** In the later stages, as the complexity of the candidate models grows, the evaluation time also increases, decreasing the genome request rates. Simultaneously, the size of the message grows and the bottleneck shifts its nature as the Manager's bandwidth saturates with throughput rather than the frequency.

B. Affected Stakeholders

The transition from a centralized to a decentralized EXAMM architecture directly addresses the constraints faced by three primary groups within the machine learning ecosystem:

- **Academic Research Laboratories:** Researchers conducting long-term neuroevolutionary experiments are the most vulnerable to the "single point of failure" inherent in star topologies. In many cases, these experiments represent weeks of nonrenewable grant-funded compute time. Neuroevolution is computationally expensive, and critically, we always start from scratch — the simplest possible networks. A single failure that stops the session can nullify all of that progress. A decentralized system ensures that a hardware glitch on a single coordinator node does not result in a catastrophic loss of progress or the corruption of the global population state.
- **Cloud Infrastructure and FinOps Teams:** There is a growing demand to optimize the use of "Spot Instances" preemptible VMs, which offer significant cost savings (up to 90%) but carry the risk of sudden reclamation [3]. EXAMM cannot be safely run on spot instances today because any reclamation kills the session. P2P architectures are naturally "preemption-proof"; because the system can self-heal and re-replicate nodes, these changes would allow running large-scale EXAMM workloads on volatile, low-cost hardware that would be unsuitable for a centralized manager-worker setup.
- **Distributed Computing Communities:** Similar to the collaborative approach seen in *Petals* by Borzunov et al. [4], there is an increasing interest in pooling heterogeneous, geographically distributed hardware. By removing the need for a high-bandwidth central authority, our approach allows smaller labs and individual contributors to participate in large-scale evolutionary training.

This work makes the following contributions to distributed neuroevolutionary computation:

- **A fully decentralized P2P architecture for EXAMM.** We redesign the centralized manager-worker topology into an asynchronous peer-to-peer ring in which every node independently generates, evaluates, selects, and migrates genomes. Rank 0 demotes itself to a standard peer after seed broadcast, eliminating the manager as both a bottleneck and a single point of failure.
- **Coordination-free distributed state via deterministic structural hashing.** We introduce a 64-bit FNV-1a hashing scheme that maps each genome's topological signature to a canonical owner node. Because the mapping is strictly deterministic, every peer independently agrees on genome ownership without a central lookup table or any coordination message, preventing redundant evaluations across the network.
- **Dual-mode genome replication.** We implement two complementary propagation protocols: MIGRATE, which routes high-fitness genomes directly to their canonical owner by structural hash, and BACKUP, which propagates the same genome sequentially to the immediate ring successor. Together they provide distributed memory consolidation and a geographic failsafe against node loss, while acting as an implicit topology-aware migration mechanism across distributed island populations. This lets the system continue to train and recover state even if $N-1$ number of nodes fail where N is the total number of peers.
- **Snapshot-based fault recovery.** Each peer continuously maintains a rolling island snapshot of its ring neighbor, updated every 10 seconds. When a peer rejoins after either a graceful disconnect or an unexpected failure, its top-k genomes are restored from the snapshot via a REJOINREQUEST/REJOINNOTIFY exchange, eliminating the cold-start penalty of single-genome reseeding.
- **Token-ring termination consensus.** We implement a distributed termination protocol in which a logic token circulates the ring and accumulates per-peer completion flags. The system shuts down only after a full token cycle confirms all peers have independently finished, removing the need for any central authority to declare session end.

- **Empirical validation under controlled fault injection.** We evaluate six experimental conditions across 54 independent trials on the NGAFID Cessna 172 aviation time-series dataset. The fault-free P2P system achieves $2.5\times$ higher genome throughput and 23% lower final MSE than Centralized EXAMM. Under improved fault tolerance, a single peer dropout stays within 18% of the fault-free P2P result and outperforms Centralized EXAMM, while three simultaneous peer dropouts — 75% of the cluster lost for over a minute — produce a final MSE comparable to the centralized baseline.

II. Related Work

Neuroevolutionary search is computationally expensive, so despite the possibility of deploying it with a single thread on a single CPU, the traditional practical approach applies distributed computation to parallelize the evaluation of individuals across a distributed cluster, which requires substantial financial investment into storage and computing resources. Converting EXAMM into a peer-to-peer system can leverage a virtually unlimited number of distributed computers to assist in evolutionary computation depending on the availability.

Systems with similar P2P architectures exist but tend to share different forms of data across one another and for different reasons than our proposed system. In the case of Petals, introduced by Borzunov et al., a similar decentralized framework is used, though in its case, for finetuning and evaluating models that are too big to fit on a single machine. In Petals, the shared model parameters are split by layers, in which each peer holds onto a different subset of layers and forwards outputs to the next peer to then be processed [4]. This system does not permit training of large models due to issues with gradient synchronization and backpropagation. Our work differs in the fact that we are not distributing model components, but instead we are distributing evolutionary genome states, which allows training of smaller models within a node. Moreover, it permits avoiding centralized coordination and improves fault tolerance.

The closest structural inspiration is a P2P genetic programming environment by Folino and Spezzano [5] that combines an island model with a cellular model in a ring topology — a different paradigm of evolutionary computation from ours, but it demonstrates that decentralized island-based evolution can achieve competitive accuracy with strong fault tolerance and virtually unlimited scalability. In our implementation we utilize a ring topology, however, instead of having cellular nodes that hold only one model, we initialize islands on each peer. While both systems use a similar form of P2P communication and decentralized ownership of model architecture data, the motivation, data being shared, programming paradigms, and recovery mechanisms remain different.

According to [2], island-model neuroevolutionary frameworks have been implemented using a peer-to-peer system and a single GPU, while the manager-worker approach is usually implemented for cluster, cloud computing, or multi-agent environments. In the parallel implementation, the classifier population and training data are divided into subsets and distributed across multiple CPUs, or "islands". Structurally, island-model neuroevolution is structurally closer to peer-to-peer, while pub/sub is operationally simpler but logically centralized (similar to the manager-worker approach). Therefore, the implementation of a mixed manager-worker island-based neuroevolutionary framework within a peer-to-peer network is a novel approach that improves fault tolerance and scalability of the existing framework by introducing variation in the choice of topologies.

A. Peer-to-Peer System Architecture

Several prior studies have explored peer-to-peer architectures for evolutionary computation, demonstrating the potential for scalability, decentralization, and efficient resource utilization. Silva et al. introduce odNEAT which is a decentralized neuroevolutionary algorithm that functions through a distributed island model where each agent holds a local population and exchanges genomes through interacting with its peers [6]. The approach involves a unilateral migration policy, meaning that genome exchange does not need a mutual agreement between two agents. Instead, each agent separately chooses whether to send out its current genome. These transmissions occur through probability rather than being set at fixed rates. The probability is based off the genomes' evaluated fitness score, so genomes that perform better than others are more likely to be shared among their peers. In the case of odNEAT, along with its fitness score, the genome also must show a 'structural innovation' that differentiates it from the agent's local population of genomes in order to be transmitted across the topology. This encourages the spread of high-performing and structurally unique solutions, introducing unnecessary redundancy.

Laredo et al. [7] propose a scalable evolutionary algorithm framework with a peer-to-peer network structure by means of a gossiping protocol enabling operations solely within the neighborhoods. This approach outperforms other distributed EA approaches reducing the computational costs on deceptive trap functions up to a few orders of magnitude. It is implemented using PeerSim Simulator [8]. PeerSim is used to evaluate performance of nodes within large-scale peer-to-peer or distributed networks and allows testing protocols of nodes interaction and program node interfaces.

A peer-to-peer implementation of the genetic programming environment for evolutionary computation is proposed by Folino et al. [5]. Their hybrid multi-island model combines an island model with a cellular model where the cellular genetic tree is evolved within an island and is migrated only to the neighboring nodes within a ring network topology. This approach achieves competitive accuracy with decentralization, enhanced fault tolerance and virtually unlimited scalability. This implementation utilizes a Juxtapose open-source P2P protocol.

Finally, Borzunov et al. present a distributed system for training large language models (LLM) Petals [4], where the peer-to-peer network is used for running inference and fine-

tuning LLMs, which allows users without direct access to large clusters to experiment with LLMs at home and contribute their idle computing resources. However, the Petals setup does not allow for distributed model training, which will be implemented within the EXAMM decentralization.

A decentralized federated learning architecture implemented by Chen et al. shifts away from the more ordinarily seen centralized server to a peer-to-peer network in which each node holds its own copy of the model and sends its local updates to be aggregated for the global model parameters, which are held across the participating peers [9]. Rather than publishing updates to a central server, nodes exchange model parameters with neighboring peers, a method that aligns with our idea of distributing genome updates among peers. Each node performs local training and works on the parameter updates it receives. Through these neighbor-to-neighbor exchanges, updates work their way through the network over time, allowing for all peers to converge without needing a single centralized aggregation point. This approach gives eventual consistency, where updates spread over time instead of being synchronized and shared all at once, which would be the case with a centralized server. Over time, the model converges through this repeated communication, showing that information such as model updates can still be shared without a manager node.

Lian et al. compare centralized parallel SGD to their proposed version of a decentralized parallel SGD (D-PSGD), using the centralized parameter-server architecture as their baseline [10]. Similar to what has been seen before, in the centralized design, all worker nodes send their updates to a single server, which holds the global model. Like the structure of the decentralized federated learning environment by Chen et al., the shift to this decentralized architecture lets each node maintain and update its own model copy while sharing updates with its peers, allowing for functionality with the entire removal of a centralized server. Without this server in the middle, a fixed stream of communication was implemented between peers where each worker only talks to its neighbors. Results show that decentralized SGD converges similarly to the centralized baseline while distributing the communication overhead across network rather than all communication being routed through a single fixed point.

B. Evolutionary Search and Island Models

Yamada et al. propose an island-model evolutionary computation system where a population is partitioned into subgroups that each evolve separately from one another, while occasionally sharing strong solutions from these subgroups through migration [11]. The system groups participants based on similarity in the solutions they create, allowing each island to focus on exploring different search directions. This would make it easier for migration policies such as the one used in odNEAT by Silva et al., to identify more meaningful changes, as opposed to minor changes, to then be transmitted across the system. When migration begins, each island sends its best-performing solutions to other islands, where they are then added to the next generation's population. Although the system is built using a centralized web architecture, it shows important island model behaviors such as adaptive grouping and migration strategies, something useful to the implementations of decentralized evolutionary systems.

Kulawik and Kruzel present EvoAVX, which is an island-based genetic algorithm library where the global population is split into multiple subpopulations that evolve separately from one another using standard genetic operators such as selection, crossover, and mutation [12]. This framework also dives into how the structure of the connection of islands and their topology, including structures like rings and grids, shape communication streams and impact evolutionary performance by influencing how information is shared between islands. Although this is implemented as a C++ library, the mechanisms and functions used for building system topology and migration strategies can be integrated into our design choices within our C++-based EXAMM environment.

C. Implementation Mechanics

Ruciński et al. show that migration topology is a first-order parameter in island-model parallel optimization, not merely an implementation detail [13]. Different topologies alter the diameter and degree of connectivity of the island network, which directly shapes how quickly genomes spread between islands and the final quality of solutions found. This motivates our choice of a structured peer-to-peer overlay in EXAMM, since the ring topology formed by consistent hashing is not arbitrary — it determines which peers exchange genomes and at what rate, making topology a design decision with direct consequences for convergence behavior.

Pavlenko et al. propose an asynchronous parallel evolutionary algorithm for gray-box optimization problems, where the fitness function for each task is broken down into multiple smaller computations [14]. Instead of forcing all worker threads to finish before moving on to the next generation, their algorithm immediately incorporates results as soon as they become available. When a worker completes its assigned task, the manager updates the population and generates new work without having to wait for the slower jobs to finish. Although this system still follows a manager-worker structure, it is shown that this asynchronous design outperforms systems that are typically synchronous in parallelization. These findings support the idea that the strategy used to coordinate tasks has positive impacts such as increases in efficiency and scalability. In our case, decentralizing the architecture by removing the manager node builds on this idea, allowing workers to operate 'freely' without scheduling constraints while improving fault tolerance at the same time.

Lombraña González et al. characterize fault tolerance in evolutionary algorithms and show that population-based search can degrade gracefully under node and task failures rather than collapsing entirely [15]. In their setting, failures appear mainly as missing results from some workers, which reduces how many candidates are available but does not stop the run. Rather than waiting for everyone, the system can move on after

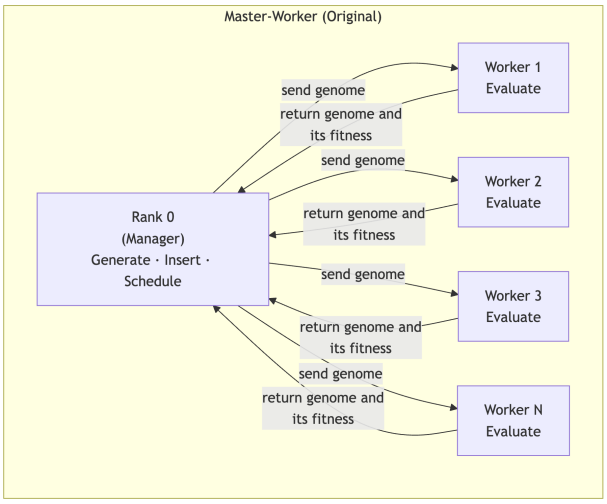


Fig. 1: Original Manager-Worker Scheme

a fixed time per iteration using whatever results have arrived, so progress continues even on slow or failing machines. This observation motivates our focus on distributing genome state and avoiding centralized coordination, since the evolutionary process can continue to make progress even when parts of the system are disrupted.

III. Methodology

To address the issues that involve scalability limits and single points of failure present in the existing centralized EXAMM framework, we propose a complete architectural redesign transitioning from a manager-worker star topology to an asynchronous peer-to-peer (P2P) ring network. This methodology details the mechanics of the structural transformation and evaluates its subsequent impact on the underlying neuroevolutionary process.

A. Transformation of Architecture to P2P

The primary goal of our design is to eliminate the centralized manager (typically MPI Rank 0) by distributing both evolutionary computation and state management equally across all participating nodes. This decentralization aims to allow for better scalability, fault tolerance, and load balancing in distributed neuroevolution systems.

1) Symmetric Node Initialization and Seed Broadcast: Symmetric Node Initialization and Seed Broadcast ensure all nodes in the P2P system begin with identical initial conditions. Unlike the previous centralized EXAMM architecture (see Figure 1), where the manager generates all genomes and dispatches them to passive worker nodes.

In our P2P implementation (see Figure 2), every compute node acts as an independent evolutionary computation engine, and every peer has its own island population of prospective genomes. To ensure that our implementation is consistent with the original work, the initialization stage remains the same. The initial population consisting of a minimal seed genome is generated on a node with rank 0, then it is serialized into a binary format and propagated to all other nodes utilizing a broadcast protocol via MPI. Once received, all nodes rewrite their output directories to rank-specific paths to prevent file system conflicts, instantiate a complete local EXAMM environment using the seed, and begin independent evolutionary loops. Rank 0 then demotes itself to a standard peer. In the pre-existing centralized EXAMM architecture, the manager generated all genomes and dispatched them to passive worker nodes.

2) Dual-Ring Topology and Active Membership: The decentralized implementation maintains two overlapping ring structures that separate failure detection from useful protocol traffic. The physical ring is fixed permanently by MPI rank order and is used for predecessor monitoring and predictable token circulation. In contrast, the logical ring is constructed from the current set of live nodes, stored as a sorted `active_ranks` vector, and is used for runtime protocol traffic including migration, backup routing, snapshot transfer, and termination forwarding.

This separation is important because the physical ring remains stable even when failures occur, while the logical ring dynamically reflects current membership. When a peer fails, it is removed from `active_ranks` and the logical ring contracts around the gap. When that peer later recovers, it is reinserted into `active_ranks`, allowing the logical ring to expand again without requiring any global reconfiguration of the physical monitoring structure.

3) Distributed State via Deterministic Structural Hashing: Distributed state management is achieved through deterministic structural hashing. It generates a unique hash value from a genome's structure, ensuring reliable mapping of genomes to their owner nodes and preventing genome identifiers' collision where a central coordinator does not exist. Figure 8 depicts the process of the token circulation between peers.

Without a central manager to track the global population and prevent redundant evaluations, we introduce this distributed memory system driver, where every node independently analyzes the topological architecture of a generated genome (its node identifiers, connection patterns, and recurrent structures) and produces a canonical string. This string is processed using a 64-bit FNV-1a hashing algorithm. The resulting hash modulo the total number of active ranks determines the "canonical owner" of that specific genome topology. Because this calculation is strictly deterministic, every node on the network inherently agrees on the storage location for any given architecture without requiring any central lookup table or coordination.

4) Asynchronous Communication (MIGRATE and BACKUP): To ensure the support for asynchronous computation as in the original EXAMM, all genome transfers are executed using non-blocking MPI primitives (`MPI_Isend`, `MPI_Irecv`). When a node evaluates a genome that represents a new global best fitness for its local environment, it replicates that genome across the network using a dual-mode communication strategy:

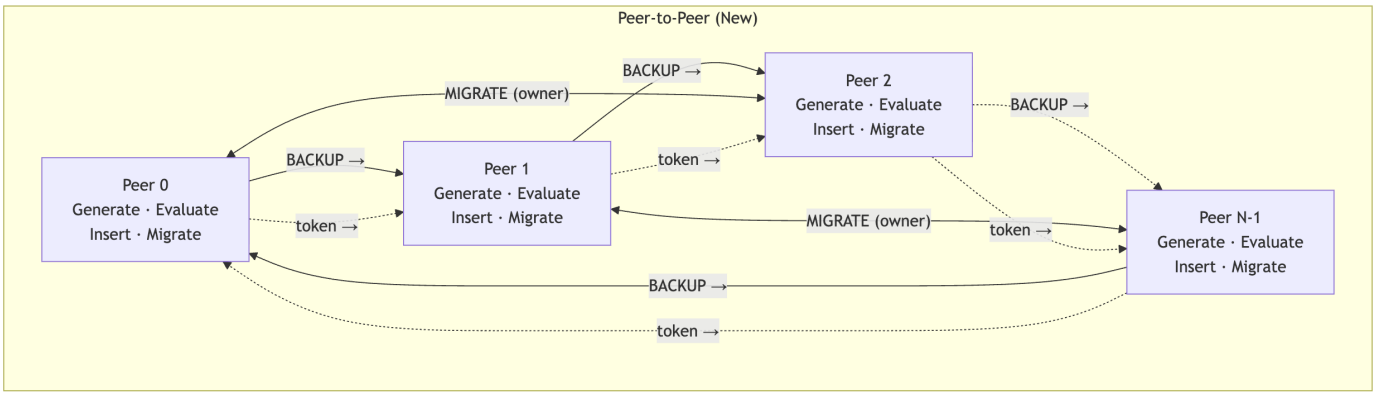


Fig. 2: Peer-to-Peer Scheme

- **MIGRATE Protocol:** The discovering node sends the genome directly to its deterministically computed canonical owner. This ensures that the node responsible for a specific topology always maintains the best-trained version of it, providing distributed memory consolidation.
- **BACKUP Protocol:** We impose a logical unidirectional ring topology across the active ranks (Rank i sends to Rank $i+1$). The discovering node sends the high-fitness genome to its immediate successor. This mechanism ensures that high-fitness traits rapidly propagate through the cluster while also acting as a secondary fail-safe against node failures.

All outgoing transfers are fully non-blocking, including both genome payloads and control traffic. Because evolved genomes can grow to tens of kilobytes as network complexity increases, large transfers are partitioned into fixed-size chunks and transmitted asynchronously so that a sender can continue local evolution while communication completes in the background. This design is essential in the decentralized setting, since a blocking send to a failed or unresponsive peer could otherwise stall progress indefinitely.

5) Failure Detection and Membership Update: The decentralized design also requires each peer to independently detect failures and update its view of cluster membership. Each node monitors only its immediate predecessor in the physical ring, while heartbeat traffic is piggybacked onto ordinary outgoing communication to the successor whenever possible. If no message has been received from the predecessor within the timeout window, the observing node declares that peer failed.

After detecting a failure, the observing node broadcasts a failure notification to all ranks in `active_ranks`. Each live peer then removes the failed rank from its local membership vector, causing the logical ring to contract around the missing node. This mechanism allows the system to converge on a consistent view of liveness without introducing any centralized coordinator or global heartbeat server.

6) Top-k State Preservation and Periodic Snapshot Push: To preserve progress under failures, each peer maintains a compact representation of its strongest local evolutionary state. Inside the island management logic, a sorted `top_k_cache` stores copies of the best genomes successfully inserted by that peer, capped at the island capacity. Because this cache is updated incrementally whenever insertion succeeds, it always contains a current summary of the peer's highest-quality genomes without requiring an expensive scan across all islands.

At fixed intervals, each peer serializes its entire `top_k_cache` into a byte buffer and sends that snapshot to its immediate right neighbor. The neighbor stores the raw snapshot bytes in a map keyed by sender rank, effectively holding a recent checkpoint of the sender's best evolutionary state. As a result, when a crash occurs, another live node already possesses a compact and recent backup of the failed peer's most useful genomes.

7) Distributed Termination via Token-Ring Consensus: A decentralized architecture lacks a singular viewpoint to determine when the entire system has converged or finished its search space. To coordinate a unified shutdown, we implemented a distributed token-ring protocol. A discrete logic token continuously circulates the logical ring. If a node finishes its local evolutionary search, it flags its completion status on the token. The system only terminates when the token completes a full cycle and confirms that all nodes have independently reported completion, ensuring no straggler operations are preemptively killed.

The token also serves as a decentralized mechanism for agreement under dynamic membership. Because the termination condition is evaluated against the current size of `active_ranks` rather than the original cluster size, consensus remains reachable even when one or more peers have failed. If the token itself is lost during a failure event, for example if it was sent to a peer that crashes before forwarding it, the system regenerates a new token after a timeout period. To avoid duplicate token creation, only the lowest-ranked surviving peer in `active_ranks` is allowed to reissue the replacement token.

8) Peer Rejoin and Snapshot-Based Recovery: When a failed peer returns, it does not restart from an empty state. Instead, it sends a rejoin request to its right neighbor in the physical ring, which is the node responsible for storing its

most recent snapshot. If a snapshot is available, the neighbor decodes the buffered genomes and returns them to the recovering peer using the same non-blocking transfer path employed for ordinary genome communication.

Recovered genomes are reinserted through the standard migrated-genome path rather than through a special-case restoration mechanism. This allows restored genomes to pass through the same fitness-gated insertion logic as any other incoming genome, preserving the normal evolutionary behavior of the system. If no snapshot is available, such as when the neighbor has also failed previously, the neighbor falls back to sending its own current top-performing genomes so that the returning peer can resume from useful genetic material rather than from a single minimal seed. After reinjection completes, the recovering peer broadcasts a rejoin notification, all live peers reinsert that rank into `active_ranks`, and the logical ring expands to include it again.

B. Impact on the Process of Neuroevolution

The shift from a population globally managed by one node to a fully decentralized system where each node holds several population islands significantly alters the dynamics of the evolutionary process.

1) Transition to an Asynchronous Island Model: In the centralized framework, the manager held a single, global population pool with a defined number of islands. Selection, crossover, and mutation depended on having a comprehensive view of all genomes. By decentralizing the system, our architecture inherently adopts a distributed Island-Model paradigm. Each peer independently performs parent selection, structural mutations (node/edge insertions), and Lamarckian weight inheritance on its isolated local population. This diversifies the population and prevents premature convergence to local minima while maximizing CPU utilization, as nodes no longer wait for a centralized queue.

2) Implicit Topology-Aware Gene Mixing: The P2P layer acts as an implicit, continuous migration mechanism. When genomes are injected from remote peers via the MIGRATE or BACKUP protocols, they are asynchronously inserted into the receiving node's local island population. Because the MIGRATE protocol routes genomes based on structural similarity (hashing), and BACKUP routes them geographically around the ring, the resulting gene pool experiences a highly diverse, topology-aware mixing. These incoming genomes are immediately eligible to be selected as parents in the local node's next generation.

3) Decoupling from the SWEET Pool: The original system utilized a centralized Selection While Evaluating in Tournament (SWEET) pool to track genomes actively being trained by workers. In the P2P conversion, the SWEET pool is strictly localized to each peer to track its own unevaluated children. Genomes arriving via network migration bypass this pool entirely, as they have already been evaluated by their originating peer. This strict separation ensures the asynchronous injection of foreign genomes does not artificially inflate the local peer's pending workload scheduler, keeping generation cycles fast and accurate.

IV. Experimental Design

In order to evaluate both the correctness and fault tolerance of the decentralized P2P EXAMM architecture, series of experiments was conducted across six conditions with nine trials each. All experiments use an identical dataset, system configuration, and fault injection schedule (only for experiments that included faults) to make sure that the differences in results stem from the condition being applied alone, instead of any variation in the environment or the workload.

A. Performance Metrics

The first metric to be evaluated is whether the decentralized architecture maintains the same level of effectiveness by comparing it directly to the original centralized EXAMM system. Specifically, we compare the best global genome fitness reached within the same time limit set to 600 seconds, including the number of genomes inserted and evaluated, and each model's fitness progression over time. By plotting these metrics against each other, we will have a clear visual on whether decentralization impacts the speed and quality of evolutionary convergence. We plan to use identical datasets and relatively equivalent configurations for each genome to ensure that the observed differences actually reflect our changes to the system architecture itself.

Fault tolerance is also measured by allowing controlled Worker failures during training and measuring metrics such as genome throughput and resulting fitness.

B. Hardware and Environment

All experiments were executed on a single machine containing an Intel Core Ultra 9 275HX processor (24 cores, 2.70 GHz base clock, 4.22 GHz boost) and 32 GB of DDR5 memory operating at 6400 MT/s, running under WSL2 on Windows. In order to manage the execution time, trials were batched three at a time and ran concurrently on the same machine. All six experimental conditions were ran on the same hardware to ensure the best comparisons upon analyzing the results.

C. Dataset

For these tests, a real-world aviation time-series dataset from National General Aviation Flight Information Database (NGAFID: C172) [16], [17] was used, which is a web portal for collecting, viewing and analyzing general aviation flight data [18]. Each trial used 32 flight sensor parameters including metrics such as altitude, airspeed, fuel amounts, and wind speeds; all as inputs to predict pitch angle as the target output variable.

D. System Configuration

Each trial instantiated 4 parallel MPI processes (peers) simulating a local P2P ring topology. Each peer maintained 10 independent evolutionary islands. The evolutionary search was bounded to a 600-second wall-clock execution limit with a maximum of 2000 inserted genomes. Backpropagation ran for 5 iterations per genome with 2 mutations applied per generation. Input parameters were normalized using min-max scaling. Candidate node types available during structural mutation included simple, UGRNN, MGU, GRU, delta, and LSTM recurrent units.

E. Fault Injection Parameters

For all fault conditions, failures were injected at t = 180 seconds into the run, with affected peers restarting at t = 240 seconds, giving a 60-second downtime window. The heartbeat broadcast interval was set to 1000 ms with a detection timeout of approximately 10 seconds, meaning surviving peers detect a failure within ˜10 seconds of it occurring. For conditions with improved fault tolerance, each peer's neighbor held onto a rolling snapshot of its island state updated every 10 seconds throughout the run.

F. Experiment Conditions

Six conditions were evaluated with nine independent trials each, totaling 54 experimental runs.

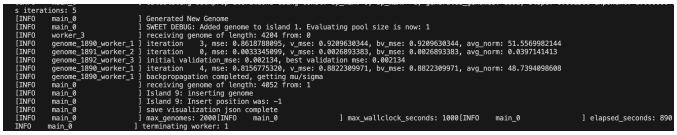
1) The centralized EXAMM baseline uses the original manager-worker architecture with no peer-to-peer communication or fault tolerance of any kind.
2) The P2P baseline runs the decentralized architecture with all four peers active for the full 600 seconds and no faults injected.

The remaining four conditions introduce controlled peer failures under two failure modes and two levels of fault tolerance.

Under **minimal fault tolerance**, the system detects peer failures via heartbeat timeout and repairs the token ring around the failed peer, the peer then rejoins, returning with no preserved state. It is then seeded with a single best genome from its neighbor and rebuilds its island population from scratch. Under **improved fault tolerance**, the same detection and ring repair mechanisms are in place, but each peer's neighbor continuously maintains a snapshot of its island state. When the peer rejoins, its full island population is restored from the most recent snapshot, which is at most 10 seconds old. If the snapshot holder itself has also crashed, the system falls back to top-k genome seeding from the nearest active neighbor.

3) One peer failure under minimal fault tolerance
4) Three simultaneous peer failures under minimal fault tolerance
5) One peer failure under improved fault tolerance
6) Three simultaneous peer failures under improved fault tolerance

In the three-peer conditions, ranks 1, 2, and 3 all fail simultaneously at t = 180 seconds, leaving rank 0 as the sole surviving peer for the duration of the downtime window.



(a) Manager-Worker EXAMM execution

(b) Decentralized P2P EXAMM execution

Fig. 3: Comparison of EXAMM execution paradigms

G. Evaluation Metrics

Each trial was evaluated on four metrics:

- Global best MSE over wall-clock time
- Global best MSE over total genomes evaluated across all peers
- System-wide genome throughput computed as a 15-second rolling window average
- Final global best MSE at the end of the 600-second run.

For comparison between trials, the mean and standard deviation of final MSE are reported across all nine trials for each condition.

Figure 3a shows the example of the evolution for the Aviation dataset with 4 parallel processes (nodes) using the original EXAMM manager-worker system architecture.

V. Results

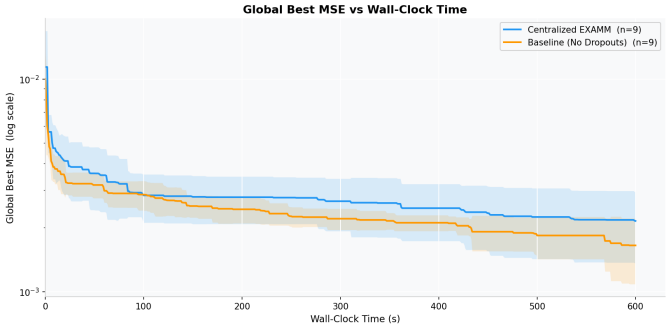
We evaluated six experimental conditions across nine independent trials each, all run on the NGAFID Cessna 172 dataset predicting pitch angle with a 600-second wallclock timespan. Faults were injected at $t = 180\,\mathrm{s}$. Table I summarizes the final mean squared error (MSE) and mean throughput for all conditions. The subsections below examine each comparison in turn.

A. Baseline: P2P vs. Centralized EXAMM

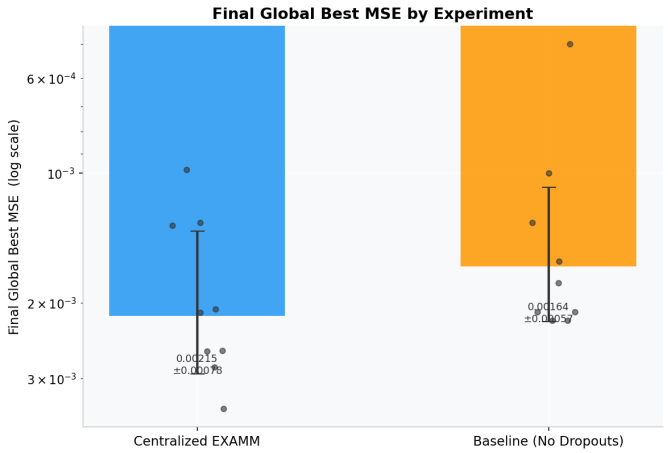
Figure 4 compares the fault-free P2P configuration against the Centralized EXAMM baseline. Across nine trials Centralized EXAMM achieved a mean throughput of approximately 1.0 genome/s and a final MSE of 2.15×10^{-3}. The P2P system, with each of its four peers holding a full island and participating in token-ring genome exchange, reached roughly 2.5 genomes/s — a $2.5\times$ throughput improvement that translates directly into faster convergence. At the end of the 600-second budget P2P EXAMM achieved a final MSE of 1.64×10^{-3}, a **23% reduction** relative to the centralized baseline.

TABLE I: Results Summary Across All Experimental Conditions (mean ± std, 9 trials each).

Condition	Peers Lost	Final MSE ($\times 10^{-3}$)	Throughput (genomes/s)	vs. P2P Baseline
Centralized EXAMM	—	2.15 ± 0.78	$\sim$1.0	+31%
P2P EXAMM (fault-free)	0	1.64 ± 0.57	$\sim$2.5	—
1-peer dropout, minimal FT	1	1.80 ± 0.59	$\sim$1.8	+10%
3-peer dropout, minimal FT	3	2.21 ± 0.36	$\sim$0.5	+35%
1-peer dropout, improved FT	1	1.93 ± 0.30	$\sim$2.0	+17%
3-peer dropout, improved FT	3	2.06 ± 0.62	$\sim$2.2	+25%

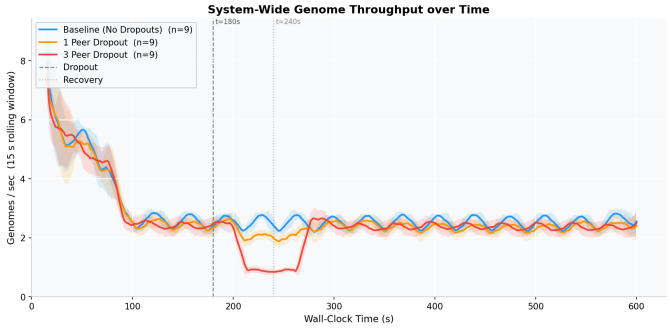


(a) MSE over time

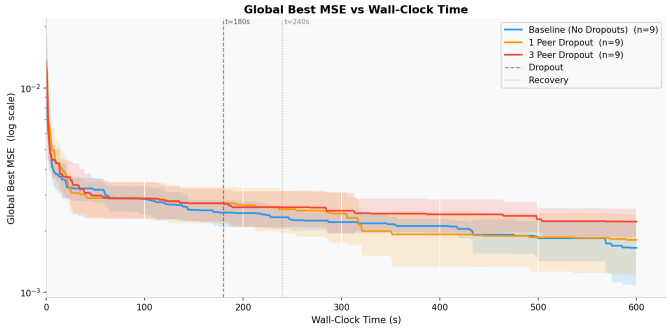


(b) Final MSE comparison

Fig. 4: Fault-free baseline: P2P EXAMM vs. Centralized EXAMM.



(a) Throughput over time



(b) MSE over time

Fig. 5: One- and three-peer dropout under minimal fault tolerance (heartbeat detection, cold rejoin).

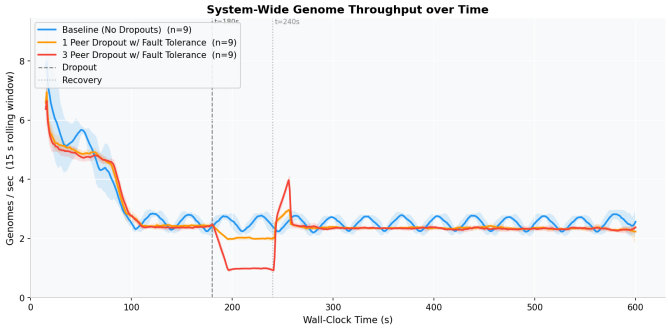
B. Minimal Fault Tolerance

Figure 5 shows peer failure under minimal fault tolerance, where the system detects a dropout through heartbeat timeout and repairs the ring, and seeds the rejoining peer with the single best genome from its neighbor. While this avoids a fully cold restart, the recovered peer must rebuild its entire island population from one seed genome, discarding the diversity of the peer that dropped out's prior search.

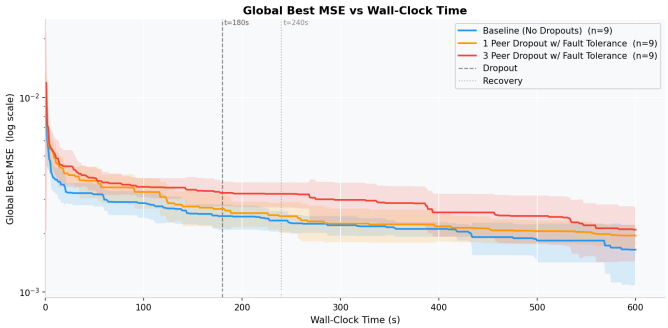
In the single-peer case the remaining three peers continue communicating throughout, the ring reroutes after the detection window, and MSE continues to improve, but the loss of island diversity on the recovered peer reduces effective search parallelism for the remainder of the run. The three peer case is a lot more severe as throughput flatlines immediately after $t = 180$ s as the token ring is severed and the lone surviving peer cannot distribute work to non-existent neighbors. Inspecting the log confirms that the `genomes_evaluated` counter across all ranks accumulates zero new entries for the full downtime window ($\sim$54 s). Even after ring repair and single-genome reseeding, three peers rebuilding from scratch within the remaining budget achieve little further convergence. These results motivate the snapshot-based improved fault tolerance evaluated in the following subsection.

C. With Improved Fault Tolerance

Figure 6 shows the same one and three peer fault conditions with improved fault tolerance in place. In both cases the surviving peers detect silence within approximately 10–11 s of the heartbeat timeout and initiate ring repair within an additional 1–4 s. Instead of reseeding with a single genome, each peer's

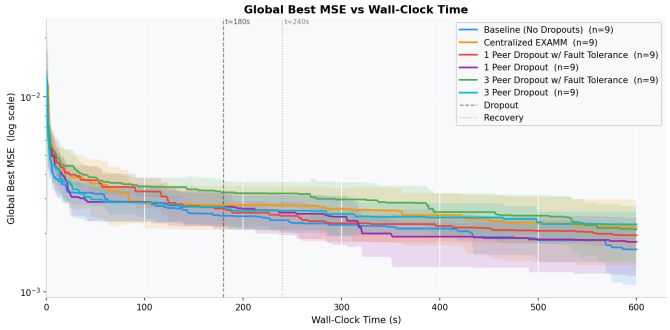


(a) Throughput over time

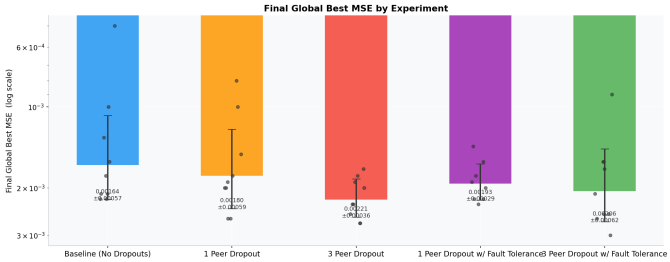


(b) MSE over time

Fig. 6: One- and three-peer dropout with improved fault tolerance enabled.



(a) MSE over time — all conditions



(b) Final MSE — all conditions

Fig. 7: All six experimental conditions compared across the full 600-second run.

neighbor continuously maintains a `peer_snapshots` map updated every 10 s. When rejoining, the failed peer receives a full island snapshot being its top-k genome population, fitness history, and structural registry through a REJOIN_REQUEST / REJOIN_NOTIFY exchange.

For the single peer case the throughput dip lasts only as long as the detection window before the ring resumes its normal oscillatory pattern, and MSE continues declining on a trajectory close to the fault-free baseline. The final MSE of 1.93×10^{-3} is 18% above the fault-free P2P result but remains **10% better** than Centralized EXAMM.

The three peer hard crash is the most adversarial condition evaluated. Rank 0 never stalls, and a log analysis of these given trials shows TOKEN_RECOVERED events firing every $\sim$10 s throughout the downtime, confirming that rank 0 cycles the token while awaiting reconnection. The throughput plot shows a very noticeable spike at $t \approx 243$ s, the moment all three snapshot-loaded peers rejoin simultaneously and immediately resume high-fitness local search. Despite losing 75% of the cluster for over a minute, the final MSE of 2.06×10^{-3} remains **comparable to Centralized EXAMM** (2.15×10^{-3}).

D. Full Experimental Comparison

Figure 7 puts all six conditions on the same plot. Table I quantifies the outcome. The key observations are: (1) the fault free P2P system sets an upper bound well above the centralized baseline; (2) improved fault tolerance recovers the majority of that advantage even under extreme fault loads; and (3) all fault conditions under improved fault tolerance remain within 26% of the fault-free P2P MSE, showing the system continues to make progress even under significant peer loss.

VI. Conclusion

Prior to this work, EXAMM existed solely as a centralized manager-worker system in which a single Rank 0 node bore full responsibility for genome generation, population state, parent selection, crossover scheduling, and session termination. This design, while operationally simple, imposed a hard ceiling on both scalability and reliability: the manager's network interface became a fan-in bottleneck as worker count grew, and any crash of the manager node terminated the entire neuroevolutionary session with no recovery path.

Our work eliminates that ceiling through a complete architectural redesign. We redesign the framework into an asynchronous peer-to-peer ring in which every node simultaneously generates, evaluates, and migrates genomes without any central authority.

P2P EXAMM achieves around 2.5$\times$ higher genome throughput and 23% lower final MSE than Centralized EXAMM from the architectural changes alone. With the implementation of improved fault tolerance, a single peer dropout stays within 18% of the baseline experiment, and three simultaneous dropouts demonstrated results outperforms those of Centralized EXAMM. The snapshot mechanism gets rid of the cold-start penalty, where recovering peers rejoin with state at most 10 seconds stale, instead of needing to rebuild from a single genome. Even the worst tested condition of three peer

dropouts under minimal fault tolerance produced a final MSE nearly indistinguishable from Centralized EXAMM.

Limitations and Future Work include: all experiments were run on a single local machine, real-world network conditions are untested. Genome payload size may bottleneck throughput, need for testing large-scale deployments. Snapshot and heartbeat interval tuning was not explored. Byzantine failures including corrupted genomes or malicious peers, not directly simulated in these experiments.

REFERENCES

[1] A. Ororbia, A. ElSaid, and T. Desell, "Investigating recurrent neural network memory structures using neuro-evolution," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 446–455. [Online]. Available: https://doi.org/10.1145/3321707.3321795

[2] F.-F. Wei, W.-N. Chen, T.-F. Zhao, K. C. Tan, and J. Zhang, "A survey on distributed evolutionary computation," *IEEE Computational Intelligence Magazine*, vol. 20, no. 3, pp. 41–62, 2025.

[3] Amazon Web Services, "Amazon ec2 spot instances for amazon emr," https://aws.amazon.com/ec2/spot/use-case/emr/, 2024, accessed: Feb. 2026.

[4] A. Borzunov, D. Baranchuk, T. Dettmers, M. Ryabinin, Y. Belkada, A. Chumachenko, P. Samygin, and C. Raffel, "Petals: Collaborative inference and fine-tuning of large models," 2023. [Online]. Available: https://arxiv.org/abs/2209.01188

[5] G. Folino and G. Spezzano, "P-cage: An environment for evolutionary computation in peer-to-peer systems," 2006.

[6] F. Silva, P. Urbano, L. Correia, and A. L. Christensen, "Odneat: An algorithm for decentralised online evolution of robotic controllers," *Evol. Comput.*, vol. 23, no. 3, p. 421–449, Sep. 2015. [Online]. Available: https://doi.org/10.1162/EVCO_a_00141

[7] J. L. J. Laredo, A. E. Eiben, M. van Steen, and J. J. Merelo, "Evag: a scalable peer-to-peer evolutionary algorithm," *Genetic Programming and Evolvable Machines*, vol. 11, no. 2, pp. 227–246, 2009.

[8] A. Montresor and M. Jelasity, "PeerSim: A scalable P2P simulator," in *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, Seattle, WA, Sep. 2009, pp. 99–100.

[9] S. Chen, Y. Xu, H. Xu, Z. Jiang, and C. Qiao, "Decentralized federated learning with intermediate results in mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 23, no. 1, pp. 341–358, 2024.

[10] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," 2018. [Online]. Available: https://arxiv.org/abs/1710.06952

[11] Y. Yamada, T. Utsumi, and K. Ohnishi, "Island model human-based evolutionary computation system," in *2025 International Conference on Machine Intelligence and Nature-Inspired Computing (MIND)*, 2025, pp. 60–65.

[12] A. Kulawik and F. Krużel, "Evoavx: Island-based genetic algorithm library with avx-512 and mpi," *IEEE Access*, vol. PP, pp. 1–1, 01 2026.

[13] M. Ruci
'nski, D. Izzo, and F. Biscani, "On the impact of the migration topology on the island model," 2010. [Online]. Available: https://arxiv.org/abs/1004.4541

[14] A. Pavlenko, D. Chivilikhin, and A. Semenov, "Asynchronous evolutionary algorithm for finding backdoors in boolean satisfiability," in *2022 IEEE Congress on Evolutionary Computation (CEC)*. IEEE Press, 2022, p. 1–8. [Online]. Available: https://doi.org/10.1109/CEC55065.2022.9870262

[15] D. L. González, J. L. J. Laredo, F. F. de Vega, and J. J. M. Guervós, "Characterizing fault-tolerance in evolutionary algorithms," in *Parallel Architectures and Bioinspired Algorithms*, ser. Studies in Computational Intelligence. Springer, 2012, vol. 415, pp. 77–99.

[16] A. P. LaBella, J. A. Karns, F. Akhbardeh, T. Desell, A. J. Walton, Z. Morgan, B. Wild, and M. Dusenbury, "Optimized flight safety event detection in the national general aviation flight information database," in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, 2022, pp. 1570–1579.

[17] H. Yang and T. Desell, "A large-scale annotated multivariate time series aviation maintenance dataset from the ngafid," 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2210.07317

[18] K. Karboviak, S. Clachar, T. Desell, M. Dusenbury, W. Hedrick, J. Higgins, J. Walberg, and B. Wild, "Classifying aircraft approach type in the national general aviation flight information database," in *Computational Science–ICCS 2018: 18th International Conference, Wuxi, China, June 11–13, 2018, Proceedings, Part I 18*. Springer, 2018, pp. 456–469.

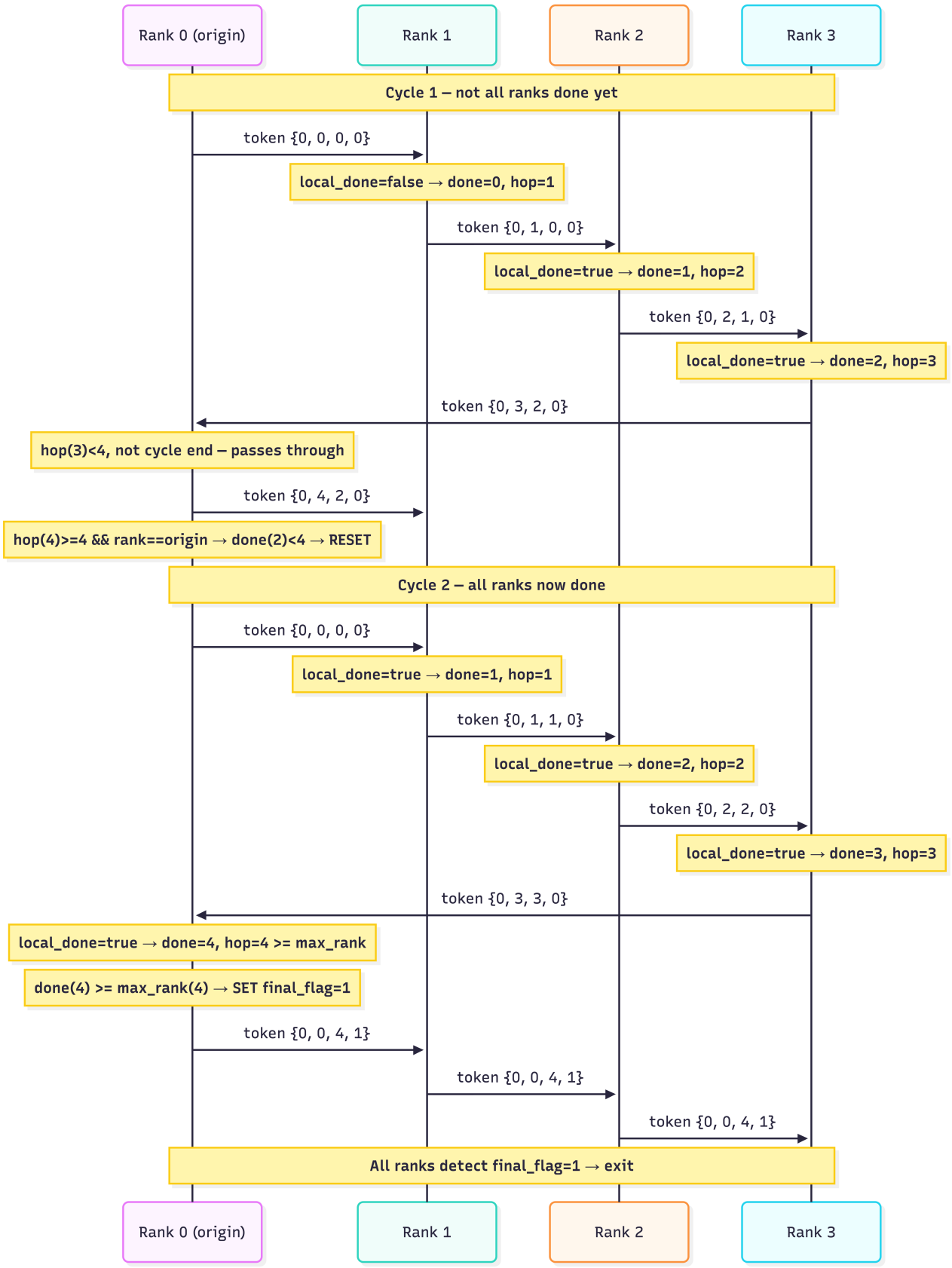


Fig. 8: Token circulation example for 4 connected peers