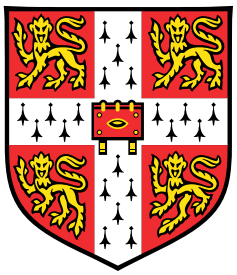
Reservoir Computing on Mobile Swarm Dynamical Systems



Yanjun Zhou

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy in Machine Learning and Machine Intelligence

St Edmund's College August 2024

I would like to dedicate this dissertation to my loving parents, I would never have made it this far without their relentless support.

Declaration

I, Yanjun Zhou of St Edmund's College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Word Count: 12850

Software Declaration: This work utilised standard Python packages including scipy, scikit-learn, and matplotlib.

A range of software libraries and repositories were also employed:

- 'boids' (https://github.com/cubeDhuang/boids) by Huang (2021) is used, with major modifications. This is originally written in java. The modifications involves converting java code to Python code, and changing the model to fit the descriptions in Section 2.4.1 and 3.1.1.

- 'PyNAnts' (https://github.com/Nikorasu/PyNAnts) by Stromberg (2021) is used, with moderate modifications to make it suitable for being utilised as the ants reservoir in our reservoir computing framework. The modification involves configuring input and output for reservoir computing, and changing the model to fit the descriptions in Section 2.4.2 and 3.1.2.

- 'echo-state-network' (https://github.com/stefanonardo/echo-state-network) by Nardo (2017) is used, with major modifications. This is originally a reservoir computing framework for ESN in MATLAB. The modifications involve converting MATLAB code to Python code, adding the observation layers, adding our swarm reservoir and adding evaluation on benchmark tasks.

The code for this project will be available upon request.

Yanjun Zhou
August 2024

Acknowledgements

I would like to express my deepest gratitude to my project supervisors, Dr. Kai-Fung Chu and Prof. Fumyia Iida, for their invaluable guidance and support throughout this project. I would also like to express my heartfelt thanks to the PhD student Mr. Fan Ye, who will become Dr. Fan Ye very soon, for helping me conquering obstacles and exploring all possibilities. The expertise and insightful feedback from all of them have been instrumental in shaping both my work and my professional growth. I am deeply grateful for the time and effort they devoted in helping me.

I am grateful to be a part of MLMI cohort, and I would like to thank all teaching and management staffs for bringing me this wonderful course. I would like to say another thank you to all the classmates in MLMI cohort, for accompanying and helping me through this unforgettable year. As one of the youngest students in the cohort, I have been receiving a lot of love and cares from you over the year.

I am thankful to every person I have encountered over the past 22 years, it is these encounters that have shaped who I am today. I thank those who once bullied me, as each setback has only made me stronger. I appreciate those who have supported me, helping me to find my way through difficult times. I thank those who once looked down on me, for teaching me to stay humble, while following my own path without being overly concerned with others' opinions. And to those who trusted me, I am grateful, as their belief gave me the courage to come this far.

Lastly and most importantly, I would like to say a special thank you to my family, I could never have come this far without your relentless support.

Abstract

Swarm intelligence, inspired by the collective behaviours of biological entities such as ant colonies and bird flocks, harnesses distributed and simple rules to address complex problems without central control. Reservoir computing enables the transformation of mobile swarm systems in real life into valuable computational resources. However, there have been limited studies in this field due to certain challenges, including permutation symmetry and instability within swarm systems, which significantly hinder the computational performance of such systems. In this work, we explore the use of highly mobile swarm systems, specifically birds and ants, within reservoir computing frameworks to perform machine learning tasks. By integrating an observation layer using Gaussian kernel density estimation into the reservoir computing framework, our approach not only addresses permutation symmetry but also stabilises swarm behaviours, resulting in a scalable swarm reservoir computer. Using four benchmark computation tasks, we explore variations in computational capacity across different swarm sizes and combinations. We have proven the effectiveness of our observation layer in solving permutation symmetry and discovered that combining different swarm reservoirs in parallel would lead to an improvement in performance. Our best reservoir model is the one that combines ants and birds reservoir at a ratio of 8:2. With a swarm size of 20, the performance achieves a covariance of approximately 0.20, comparable to that of ESN16. As the number of swarms increases to above 60, the covariance value reaches around 0.21, matching the performance of ESN18. This indicates that our swarm reservoir has a reasonable amount of memory and nonlinearly capacity in performing computation tasks. Our findings also delve into the impacts of altering system parameters on the networks' computational abilities, offering insights into mechanisms in swarm intelligence and application to AI.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Swarm intelligence is a field of study that draws its principles from the collective behaviours observed in natural systems such as ant colonies, bee hives, and bird flocks. These biological systems exhibit a remarkable ability to solve complex problems through simple interactions between individuals without central control (Deneubourg and Goss, 1989; Emlen, 1952; Lissaman and Shollenberger, 1970; Mataric, 1992). This distributed decision-making process allows swarms to adapt dynamically to environmental changes, optimise routes, and even construct complex structures (Kvam and Kleiven, 1995; Sumpter, 2010).

In the context of computational neuroscience and artificial intelligence, swarm intelligence has been used to inspire the development of robust, scalable networks capable of solving real-world problems. This includes the SpikeAnts model proposed by Chevallier et al. (Chevallier et al., 2010). These artificial neural networks often treat swarm agents as nodes within the network, with the edges representing interactions between these nodes. Each node in the network acts independently, following simple rules based on the local information it perceives. This leads to the emergence of coordinated group behaviour as a global pattern. The connections between nodes change as the swarm system progresses. Mobile networks of swarms evolve their architecture autonomously based on inherent dynamics, without the need for explicit programming or training. Therefore, the research of swarm intelligence contrasts traditional evolutionary neural networks (ENNs), which use evolutionary computation to optimise neural architectures (Stanley et al., 2019; Stanley and Miikkulainen, 2002; Wierstra et al., 2014).

Utilising swarm intelligence for computation transforms mobile swarm systems in real life into valuable computational resources. This encompasses mobile vehicles and artificial robot swarms, which are highly mobile and consist of similar agents. These systems offer significant scalability and flexibility, providing substantial computational power that contributes to societal advancement.

Reservoir computing (RC) offers a unique approach to computational intelligence, which allows the inherent dynamism of swarm systems to be harnessed to perform computations (Jaeger, 2001; Jaeger and Haas, 2004). In RC, the swarm networks act as "reservoirs" that process input signals through their natural dynamic responses. The core of reservoir computing is its ability to capture these dynamic reactions, which are rich in time-based and spatial information, and convert them into practical computational outputs. This approach leverages the inherent capabilities of swarm systems to efficiently manage complex computing tasks.

There has been very limited research on harnessing the collective intelligence of highly mobile swarm systems composed of similar agents. These systems are generally difficult to control and are highly prone to permutation symmetry, which comes from the swarm agents swapping positions with each other (Lymburn et al., 2021). These issues all lead to reductions in computation capacity, bringing challenges for utilising swarm systems for computation. The research conducted by Lymburn et al. (2021) utilises the modified Reynolds boids model to explore swarm responses to predator-like signals in nonlinear time-series prediction tasks. They tackle permutation symmetry using a radial basis-localised observation layer. However, the interactions between predator and prey, and those among animals within the same species differ significantly, indicating that the intelligence observed is not merely attributable to a single animal swarm.

In this work, we leverage the self-organising and adaptive learning capabilities of biological swarms with semi-homogeneous agents, and integrate them into reservoir computing frameworks to perform machine learning tasks. Two types of swarm systems are utilised in this study, namely boids (bird-oids) (Reynolds, 1987) and ants (Beckers et al., 1990; Dussutour et al., 2009; Grüter et al., 2011). Similar to the work of Lymburn et al. (2021), we address permutation symmetry by incorporating Gaussian kernel density estimation into the readout of swarm reservoirs, which produce output based on the overall distribution of reservoir states. Different from them, we examine how computation power evolves from individual agents to collective swarm intelligence, and evaluate the variation in computation capacity for swarm systems of different sizes. Then, different swarm systems are combined to see if we can harness the advantages of different systems arising from different dynamics. The impact of altering system parameters on the computational capacity of these networks is also investigated. The benchmark datasets we use for performance evaluation include short-term memory (STM), parity check (PC), nonlinear autoregressive moving average (NARMA) and Markey-Glass (MG) time series prediction (Jaeger and Haas, 2004). Although our study is based on simulations of swarm systems, it can provide valuable insights into physical implementations of reservoir computing with swarms in future studies.

1.1 Main Contributions

The main contributions of our work are summarised as below:

- We propose swarm reservoir designs based on simulation models of birds and ants, which are highly mobile swarm systems composed of semi-homogeneous agents. We select suitable input and output for both swarm systems, to make them function reservoirs with computation capacity.
- We address the permutation symmetry issue in swarm reservoirs with Gaussian kernel density estimation, to produce output based on the distribution of states.
- We investigate the effect of combining different swarm reservoirs in parallel and witness improvements in computation capacity.
- We conduct extensive experiments to evaluate the computation capacity of our framework on four benchmark datasets. We have also compared the performance of our swarm reservoir with one of the most popular reservoirs, Echo State Networks (ESN). Experiment results have shown that the computation capacity of our combined swarm reservoir is equivalent to ESN16 at a swarm size of 20. As the swarm size becomes greater than 60, the performance attains a covariance of around 0.21, similar to ESN18.

1.2 Dissertation Outline

The dissertation is organised as follows:

- **Chapter 2** reviews the related works in reservoir computing, describing how the reservoir computing framework enables physical objects or swarms to be utilised for computation. The chapter also discusses the swarm models that we use in this work.
- **Chapter 3** describes the reservoir computing framework we use, including the formulation of using swarm systems for reservoir computing.
- **Chapter 4** presents and discusses the experiments and results on benchmark computation tasks
- **Chapter 5** concludes the paper and presents the future directions

Chapter 2

Background

This chapter outlines the theoretical foundations and related works that support our research. Section 2.1 describes the classical reservoir computing (RC) framework using virtual reservoirs. Then Section 2.2 describes how the classical RC framework enables physical robots to be utilised for computation, including some successful previous works. Section 2.3 discusses the existing studies of utilising swarm intelligence in RC, highlighting the research gap that our work addresses. Subsequently, Section 2.4 presents details of the swarm models that we use in this work. Finally, Section 2.5 summarises the key points of this chapter, including the problem that our work addresses.

2.1 Classical Reservoir Computing Framework

Reservoir Computing (RC) is an innovative computational approach within the broader field of artificial neural networks, specifically designed for processing time series and dynamic data. Echo state networks (ESNs) are one of the pioneering and classical reservoir computing methods that was proposed, drawing inspiration from recurrent neural networks (RNNs) (Jaeger, 2001; Jaeger and Haas, 2004). Figure 2.1 shows the comparison between typical RNNs and ESNs. Both architectures consist of interconnected neurons that form directed cycles, allowing them to maintain an internal state that acts as a memory of previous inputs. However, the two networks differ in their training approach. RNNs require training across all network weights, including the input layer, hidden layer and output layer. The training involves iterative methods to optimise the weights that minimise the prediction error. By comparison, ESNs use a random fixed set of internal weights for the input and hidden (reservoir) layer, and only train the output (readout) weights. As weights of both the input and reservoir layers are fixed, the input to the readout layer would always remain consistent

under the same input. Therefore, single-epoch training using linear regression is adequate for training the output layer weights, with no need for iterative adjustments.

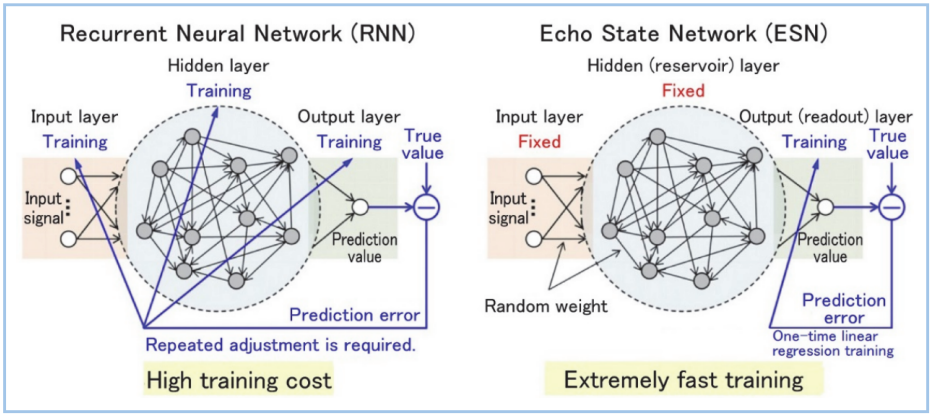


Fig. 2.1 Comparisons between recurrent neural network (RNN) and echo state network (ESN) from TAKAGAKI et al. (2024)
. Traditional RNN trains all network weights using iterative methods, while ESN only trains the weights in output (readout) layer with single-epoch training.

The reservoir computing approach of ESNs provides a notable advantage over traditional RNNs by significantly streamlining the training process while maintaining the capability to model complex time series. This streamlined method not only speeds up training but also cuts down on computational requirements, making ESNs exceptionally efficient. Unlike RNNs, which use resource-intensive backpropagation through time (BPTT) to train all network connections, ESNs concentrate training solely on the output layer. This approach not only circumvents the vanishing gradient issue but also minimizes the amount of data required for effective training. Consequently, ESNs are especially effective for managing large-scale or intricate time series data, ensuring rapid and dependable model training.

In the domain of reservoir computing, the hidden layer of the reservoir computing framework in Figure 2.1 is commonly referred to as the reservoir, the outputs derived from the reservoir are termed reservoir states, and the output layer is denoted as the readout. These terminologies will be consistently employed throughout this thesis.

As the reservoirs of ESNs use the recurrent architecture like RNNs, the reservoir states are computed from input in a similar manner (Lukoševičius, 2012). At each time step n, the reservoir state $x(n) \in \mathbb{R}^{N_x}$ is updated based on the input $u(n) \in \mathbb{R}^{N_u}$ and its previous state $x(n-1) \in \mathbb{R}^{N_x}$ using the equation:

$$\hat{x}(n) = \tanh(\mathbf{W}^{in}[1; u(n)] + \mathbf{W}x(n-1)), \tag{2.1}$$

where $\hat{x}(n) \in \mathbb{R}^{N_x}$ represents the update in state, and $\mathbf{W}^{in} \in \mathbb{R}^{N_x \times (1+N_u)}$ and $\mathbf{W} \in \mathbb{R}^{N_x \times N_x}$ are the input and reservoir weight matrices, respectively. The function $\tanh(\cdot)$ is applied element-wise, and $[1; u(n)]$ denotes the vertical concatenation of 1 and the input vector $u(n)$. The state $x(n)$ at any time step is a linear interpolation of the previous state $x(n-1)$ and the updated state $\hat{x}(n)$, calculated as:

$$x(n) = (1-\alpha)x(n-1) + \alpha\hat{x}(n) \tag{2.2}$$

where $\alpha \in [0,1]$ denotes the leaking rate, which controls the rate at which the reservoir updates its state. The readout weights $\mathbf{W}^{\text{out}} \in \mathbb{R}^{N_y \times (1+N_u+N_x)}$ are trained through linear regression or ridge regression, which will be explained in detail in Section 3.3 The network output $y(n) \in \mathbb{R}^{N_y}$ from the readout layer is computed such that

$$y(n) = \mathbf{W}^{\text{out}}[1; u(n); x(n)]. \tag{2.3}$$

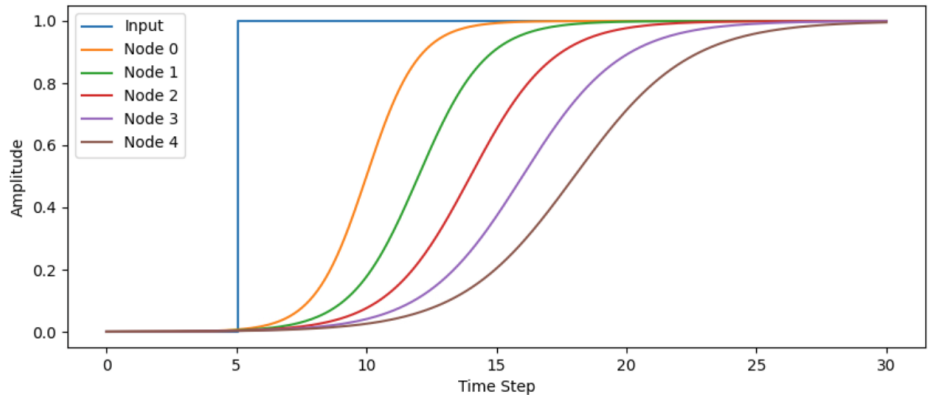


Fig. 2.2 Example of echo state property under a step input (blue). Different nodes respond to input at different delays, and delay results in the memory of previous states. The nodes with longer delays would have memories that last longer.

From Equation 2.2 and 2.1 above, it can be seen that the current state of ESNs is dependent on both current input and previous states. For an ESN node x with large reservoir weights $\mathbf{W_x}$, small leaking rate α and input weights $\mathbf{W_x}^{in}$, the current state would be highly dependant on previous states. This means that the node is having a memory of previous states, that gradually fades with time. Otherwise, if the reservoir weights $\mathbf{W_x}$ are small, while the leaking rate α and input weights $\mathbf{W_x}^{in}$ are large, the node would follow the input more tightly with memory fading faster. For a step input applied to an ESN reservoir, the nodes would respond to input with different delays, some nodes have a fast response to input changes, while others hold a longer memory of past states. This response is illustrated in Figure

2.2, and is referred to as echo state property in previous studies (Jaeger and Haas, 2004). The computation capacity of a reservoir computer generally includes memory capacity and nonlinearity capacity. The 'echo states property' produces memory capacity within ESNs, while the nonlinearity capacity comes from the rich dynamics within the complex and random connections of the reservoir.

Later studies have shown that the reservoir computing approach can be extended to using physical objects as reservoirs, which involves replacing the reservoir of ESN with physical objects (Baldini, 2022; Nakajima, 2020; Tanaka et al., 2019). A suitable set of input and output are needed to be selected for the physical reservoir so that the output exhibits echo state property and rich nonlinear dynamics.

2.2 Reservoir Computing with Single Robot Reservoirs

ESNs use virtual reservoirs of randomly generated RNNs, which still require the usage of computational resources. The computation capacity of ESNs is also inconsistent for different sets of random architectures. By comparison, physical reservoirs utilise the natural dynamics of a physical medium, conducting computations through the inherent physical properties of the system. The approach not only reduces the amount of computation resources required, but also makes the computation capacity more consistent as physical reservoirs generally have relatively fixed structures. Physical reservoirs can also be used in the development of embodied intelligence, enabling the robot to be driven by itself.

2.2.1 Octopus-inspired Soft Robotic Arm

Several studies have investigated the application of a soft robotic arm inspired by octopus morphology as a physical reservoir in reservoir computing (Nakajima et al., 2013, 2015, 2014). By using the nonlinear dynamics of the robotic arm's soft structure, the research showcases its capability to approximate complex systems and perform computation tasks.

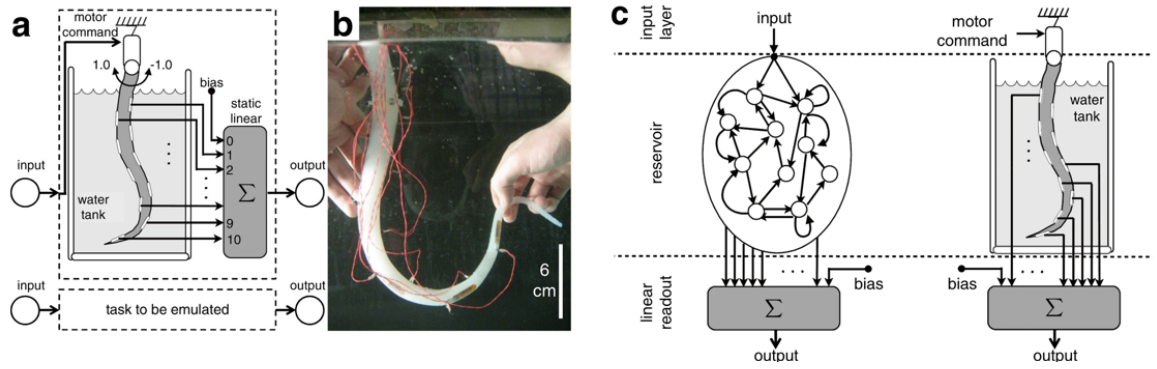


Fig. 2.3 Setup of a platform for a soft silicone arm, along with diagrams illustrating the information processing framework employed by the arm, from Nakajima et al. (2015). (a) Schematic of the physical reservoir computing framework, with the silicon arm as a reservoir. The silicon arm is placed in water, with 10 bend sensors attached to it. The input is injected from the motor on the top, inducing bends that propagate along the arm. (b) Picture of silicon arm used in their study. (c) Comparison between conventional reservoir computing framework and physical reservoir computing framework in this study. The virtual reservoir is replaced by the silicon arm, which exhibits complex dynamics that produce memory and nonlinearity capacity.

The structure of this reservoir is shown in Figure 2.3a and 2.3b. A motor is attached to the soft robotic arm, to make it swing leftwards and rightwards. The soft robotic arm is placed underwater, with 10 bend sensors attached to it, to measure the bending in different parts of the arm. If the bend occurs on the ventral side, the sensor value falls below 1. Conversely, if the bend is on the dorsal side, the value exceeds 1. For this reservoir, motor rotation is selected as the input, and the bending measured by sensors is chosen as the output. When the motor at the top rotates, it induces a bend that propagates along the soft robotic arm and gradually diminishes as it approaches the end. This satisfies the echo state property mentioned in Section 2.1, making this soft robot a suitable reservoir. Figure 2.3c shows the comparison between this system and conventional reservoir computing systems like ESNs. It can be seen that the virtual reservoir network is now replaced by this physical reservoir.

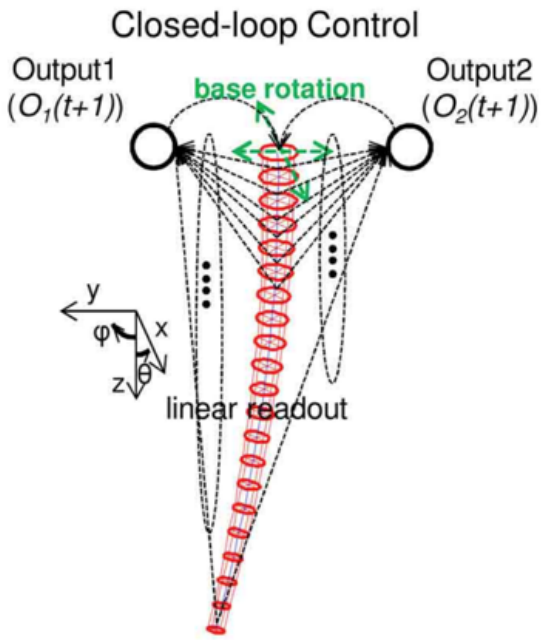


Fig. 2.4 Schematic diagram of closed-loop control for silicon arm from Nakajima et al. (2013). This self-driven system is able to emulate predefined movements without external inputs. The output is now fed back to the input, and the readout is trained to map the output to the target input.

This physical reservoir computing system demonstrates strong performance in short-term memory (STM) tasks and nonlinear auto-regressive moving average (NARMA) tasks, thereby validating its computational capabilities. The system is also extended into a reservoir controller that drives the robotic arm itself, making it execute a predefined movement. This is done by closing the loop of the system in Figure 2.3c, making the output to be the input of the next time step. This is illustrated in Figure 2.4. The training phase is carried out in an open loop, where the readout weights are trained to map the reservoir output to the target input of the next time step. Then the evaluation phase is carried out in a closed loop, with the system driven by itself without external input. The predefined movements have been successfully replicated, demonstrating the effectiveness of this control approach.

2.2.2 Spine-Driven Quadruped Robot

The study by Zhao et al. (2013) explores the utilisation of spinal dynamics as a computational resource in a spine-driven quadruped robot. It applies reservoir computing to develop a self-driven robot, showing the ability of the spine to adapt and execute various locomotive behaviours like bounding, trotting, and turning.

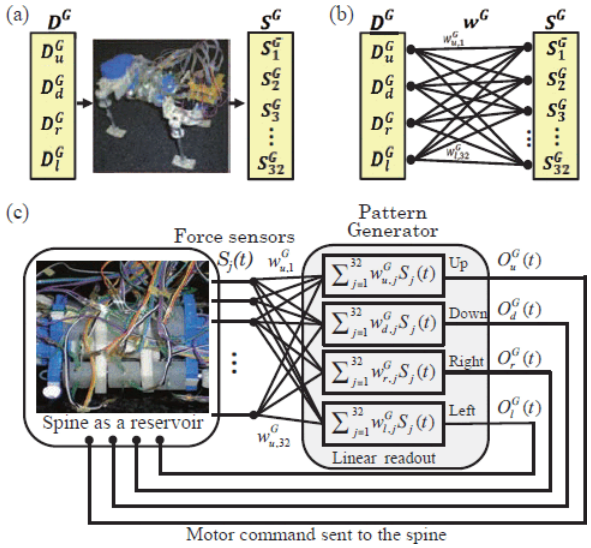


Fig. 2.5 Schematic diagram of reservoir computing with spine-driven quadruped robot from Zhao et al. (2013). (a) The spine reservoir takes motor command $\mathbf{D}^G$ as input, with output $\mathbf{S}^G$ collected from 32 force sensors. (b) Readout weights $\mathbf{w}^G$ trained in an open loop to fit the target output. (c) Schematic of the closed loop system, with the pattern generator computing input of next time step from output and readout weights.

Figure 2.5a shows the robot reservoir that they used. The spine of the robot is controlled by 4 motors attached to it, and the movement of the robot is driven by the movement of the spine. The input to the reservoir $\mathbf{D}^G$ is the motor command, and the reservoir output $\mathbf{S}^G$ is the readings from 32 force sensors along the robot spine. An input to the reservoir would create a force that propagates along the spine. This means that the sensors would respond to the input with different delays, making the reservoir exhibit echo state property and memory capacity. The nonlinearity capacity comes from the complex dynamics of the spine, enabling it to manage complex control tasks effectively. Figure 2.5b shows how the readout is trained for the robot to perform a predefined task, with the target motor command of performing the task provided. The training stage is carried out in an open loop, to train the readout weights $\mathbf{w}^G$ that maps sensor output $\mathbf{S}^G$ with the target motor command $\mathbf{D}^G$. Figure 2.5 shows the evaluating phase after training, The control loop is now closed, with a pattern generator computing motor command based on the reservoir output $\mathbf{S}^G$ and readout weights $\mathbf{w}^G$. The control command is then sent back to the input of the reservoir (spine). The study has demonstrated that the spine reservoir are able to drive the robot to emulate different types of gaits, proving the effectiveness of this control approach.

2.3 Reservoir Computing with Swarm Reservoirs

Utilising swarm intelligence for computation has been successful in developing optimisation algorithms. Particle Swarm Optimisation (PSO) (Kennedy and Eberhart, 1995) mimics the social dynamics observed in birds and fish. It utilises a swarm of particles that adjust their positions based on personal and collective achievements, efficiently balancing exploration and exploitation in the solution space. This simplicity and minimal parameter tuning make PSO ideal for continuous optimisation problems where rapid convergence is desired. Ant Colony Optimisation (ACO) (Dorigo et al., 2006) draws inspiration from the foraging behaviour of ants, using artificial ants that traverse a problem graph while depositing pheromones to guide others towards promising solutions. This method excels in discrete optimisation challenges like routing and scheduling, benefiting from a robust mechanism that adjusts to dynamic environments and helps avoid local optima through pheromone evaporation. Both algorithms provide rapid convergence to optimal solutions, especially in complex optimisation problems. This performance is attributed to the flexibility and scalability inherent in swarms.

Looking at the application of swarm intelligence in reservoir computing, swarm reservoirs are typically more flexible when comes to scalability, as the size of network can be easily changed by having individuals joining into or leaving out of the swarm. On the contrary, the single robot reservoirs in Section 2.2 might suffer from scalability issues, as an increasing number of nodes might involve physical modifications to the robot itself. Another difference between swarm reservoirs and other reservoirs is that the interconnections between reservoir nodes change with time, as the amount of interactions between swarm agents varies as the swarm travels. This evolving architecture could potentially lead to high flexibility and adaptability to different inputs. Additionally, reservoir computing with swarms may also be used for the development of embodied intelligence for artificial swarm systems. By integrating reservoir computing, artificial swarms can process environmental inputs in real time. This capability not only enhances their decision-making processes but also improves their ability to handle complex tasks in dynamic environments.

2.3.1 Particle Swarm

Research from Wang and Cichos (2024) have shown the effectiveness of harnessing particle swarms for reservoir computing, where these particles dynamically respond to external stimuli through delayed propulsion mechanisms. Similar to Section 2.2, they used particle swarm reservoir to emulate a self-driven system performing Markey-Glass tasks (Wang and Cichos, 2024).

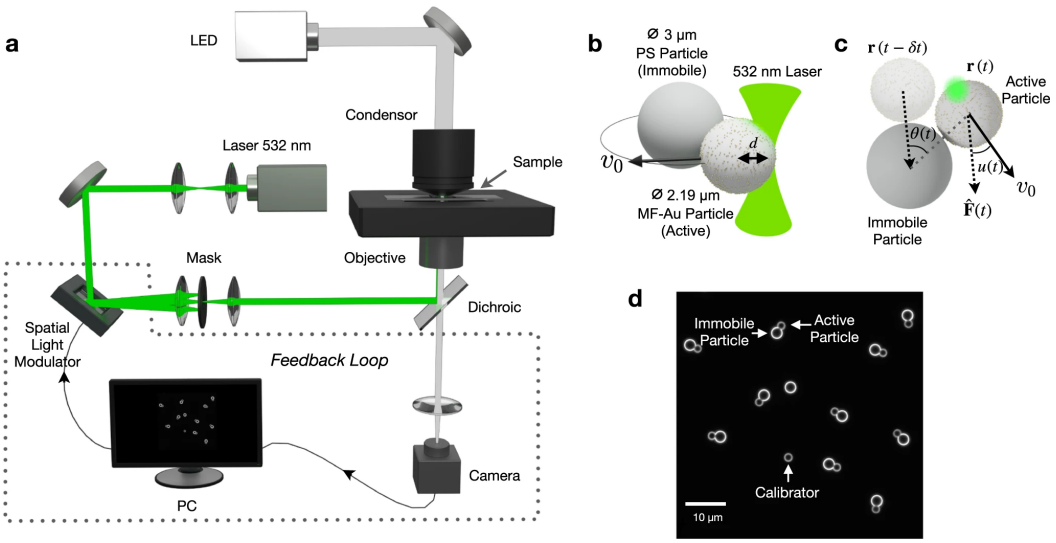


Fig. 2.6 Experimental realisation for reservoir computing with particle swarm from Wang and Cichos (2024). (a) Experimental setup, with particles in the sample heated by the laser (532 nm), illuminated with LED light and captured by the camera for analysis on PC. (b) Mobile and Immobile particles in the sample. (c) Heating particles by laser would induce force $\hat{F}(t)$, making the particle to rotate around the immobile particle by an angle of $\theta(t)$. (d) Image of the sample, with an additional calibrator is used adjust the particles' movement.

As shown in Figure 2.6b and 2.6d, the particle swarms are consisted of immobile PS particles and mobile MF-Au particles. Figure 2.6c shows that heating the particles with a laser would induce a force $\hat{F}(t)$ onto the active particle, making it rotate around the immobile particle by an angle of $\theta(t)$. The orientation angle θ of the particle responds to laser input with a delay, hence satisfies the echo state property. Therefore, it is chosen as the output for the swarm reservoir. Figure 2.6a shows the layout of their setup, where the sample of MF–Au microparticles are heated by a beam from 532nm laser. The input from the laser to the swarm reservoir is adjusted by the spatial light modulator, to produce an input that varies with time. The sample is illuminated with LED light, and a camera captures the states of the swarm reservoir for analysis on a PC. The readout trained from the PC converts the reservoir states to the commands to a spatial light modulator, which adjusts the input to the reservoir. The training and evaluation methods are in similar manners to the self-driven physical reservoirs in Section 2.2, with readout training being carried out in open loop, matching reservoir states to target reservoir input.

The reservoir obtained precise predictions in Markey-Glass prediction tasks. However, the particle swarms that they use are in relatively fixed positions in space, and the variation in interconnections between reservoir nodes is not that significant. By comparison, our study

will explore the use of highly mobile swarm agents, with network interconnections changing significantly over time.

2.3.2 Predator-Driven Boids

The study by Lymburn et al. (2021) leverages the modified Reynolds boids model as a reservoir to investigate how swarms respond to a predator-like signal for nonlinear time-series prediction tasks. The boids model simulates the behaviour of birds in nature, including separation, alignment, and cohesion. These behaviours will be discussed in detail in Section 2.4.

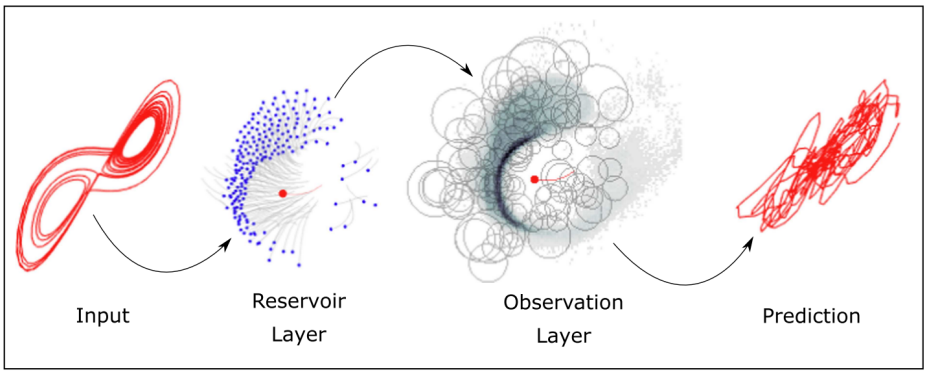


Fig. 2.7 Schematic of reservoir computing framework with predator-driven boids reservoir, from Lymburn et al. (2021). The input is injected by the movement of the predator (red dot), driving all the other birds to flee from it. The observation layer records the overall shape of the swarm, making the output invariant of internal permutations among individuals. This output is then used for performing Lorenz prediction tasks.

The reservoir computing framework utilised in this study is shown in Figure 2.7. In addition to the basic boids model, a predator is added in this study, and its trajectory of movement acts as an input to swarm reservoir. In the reservoir layer part within the figure, the predator is shown as a red dot and the prey are shown in blue. When the predator forages, the prey would flee from the predator at different delays, making the positions of prey exhibiting echo state property suitable for reservoir output.

The issue of permutation symmetry in highly mobile swarm systems is discussed in this study. When swarm trajectories approach a previously encountered state, the same input drive patterns can lead to diverse responses if the positions of individual agents are altered or swapped. The behaviour of an individual bird is significantly influenced by interaction with its neighbours. If the bird is in a different position under the same input, it would respond differently as it is interacting with a set of different neighbours. This causes the swarm

behaviour to differ under the same input applied, and this inconsistency makes it difficult to train the readout weights. This issue is resolved by observing the overall shape of the swarm, shown by the observation layer part in Figure 2.7. By producing output based on the overall shape, the result would become invariant of internal permutations.

The solution draws inspiration from the radial basis function modelling technique (Lowe and Broomhead, 1988) by positioning Gaussian observation kernels on a random set of M agents across the swarm's spatial domain. These kernel functions are defined as follows:

$$\psi_m(x) = e^{-\frac{(x-c_m)^2}{2w_m}} \tag{2.4}$$

where $m = 1, 2, \ldots, M$ enumerates the agents. The centre of kernel c_m is defined by the agent's position, and kernel width w_m is set as the distance to the fifth nearest neighbour of the central agent. Observations are computed by applying the kernels to the velocities and positions of the birds:

$$r_{1,m}(t) = \sum_{i=1}^{N} \psi_m(x_i(t)) \tag{2.5}$$

$$r_{2,m}(t) = \sum_{i=1}^{N} \psi_m(x_i(t)) v_{xi}(t) \tag{2.6}$$

$$r_{3,m}(t) = \sum_{i=1}^{N} \psi_m(x_i(t)) v_{yi}(t) \tag{2.7}$$

Equation 2.5 quantifies the number of agents within the field of the kernel, and then Equations 2.6 and 2.7 compute the average velocity of birds near the centre of the kernel. Together, these metrics capture the motion of the swarm across different spatial locations, forming the circles in the observation layer part of Figure 2.7. Then different circles would superpose with each other to form the overall shape of the swarm. This is achieved by concatenating the data from all 3M kernels to create the output of the observation layer (shape of swarm), which is then fed to the linear readout.

This framework performs reasonably well in Lorenz prediction tasks. However, this system needs to be driven by a predator, and the performance is also highly dependent on the predator's performance. In this study, we will utilise a self-organised swarm system with all the agents having the same interactions, therefore they are equivalent to each other, being a practical swarm. We will also conduct a more thorough investigation of the computation capacities in using more benchmark tasks, in both open-loop and closed-loop (self-driven) frameworks. Although our study is based on simulations of swarm systems, it can provide

valuable insights into physical implementations of reservoir computing with swarms in future studies.

2.4 Swarm Systems

This section describes two types of swarm models, namely Boids model and Ants model. Both models are multi-agent systems mimicking the behaviours of swarms. We will dive into the internal dynamics of the models, including how interactions between swarm agents take place.

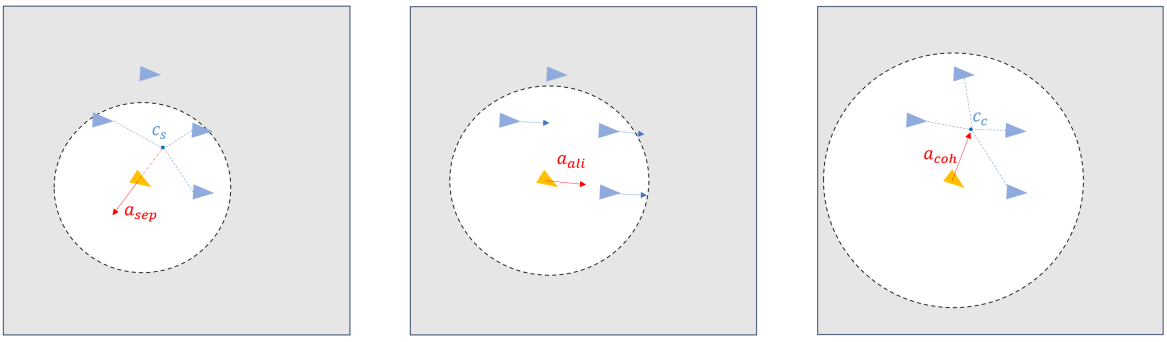
2.4.1 Boids Model

The Boids model developed by Reynolds (1987) operates on a combination of simple behavioural rules applied uniformly to each agent within a simulated environment. Each single agent in the model is driven by three types of force factors: alignment, cohesion and separation. The total force on a single bird would be the weighted sum of these three forces.

$$F_{total} = k_{sep}F_{sep} + k_{ali}F_{ali} + k_{coh}F_{coh} \tag{2.8}$$

where k_{sep}, k_{ali} and k_{coh} are factors that weights the three forces. Assuming all the birds in the system have the same mass m, the acceleration can be updated such that

$$a_{total} = k_{sep}a_{sep} + k_{ali}a_{ali} + k_{coh}a_{coh} \tag{2.9}$$



(a) (b) (c)

Fig. 2.8 (a) Separation rule makes the birds move away from the centre of mass of neighbours within the separation radius. (b) The alignment rule makes the velocity of birds match with its neighbours within the alignment radius. (c) The cohesion rule makes the birds move towards the centre of mass of neighbours within the cohesion radius.

Figure 2.8a illustrates the rule of separation among the birds, with the bird in orange moving away from the centre c_s of its neighbours. Assuming that for a single agent j, there are N_s number of other agents in its visual range R_s for separation, then the position for the centre of mass c_s of these agents can be calculated as

$$(x_{c_s}, y_{c_s}) = \frac{1}{N_\mathrm{s}} \sum_{i}^{N_\mathrm{s}} (x_i, y_i) \tag{2.10}$$

Here, (x_i, y_i) represents the position of the i-th bird in the separation range, which satisfies $\left\|(x_i, y_i) - (x_j, y_j)\right\| < R_s$. Then the sum is calculated over all birds within the range. The principle of separation within a flock serves to prevent crowding, by maintaining a minimum distance between each bird and its nearby flockmates. This is achieved by adjusting the acceleration of each bird so that it moves away from the centre of mass of its neighbours. The acceleration for bird j, therefore, is updated according to the following formula:

$$a_{sep} = (a_{x_j}, a_{y_j}) = (x_j, y_j) - (x_{c_s}, y_{c_s}) \tag{2.11}$$

The sum is calculated over all birds i that are different from bird j, effectively pushing bird j away from areas of high density and helping to maintain an optimal spacing within the flock.

Alignment ensures that birds within a flock head in the same general direction, where each bird adjusts its steering towards the average direction of its nearby flockmates. This rule facilitates synchronised movement across the flock and is illustrated in Figure 2.8b. For each agent in the system, the adjustment in acceleration is calculated as the sum of the differences in velocities between itself and each nearby bird. Assume that there are N_a birds in the alignment range R_a of bird j, the calculation for the acceleration component can be mathematically represented as:

$$a_{ali} = (a_{x_j}, a_{y_j}) = \sum_{i}^{N_\mathrm{a}} \left[(v_{x_j}, v_{y_j}) - (v_{x_i}, v_{y_i})\right] \tag{2.12}$$

Here (v_{x_j}, v_{y_j}) represents the velocity of bird j and (v_{x_i}, v_{y_i}) represents the velocity of bird i in the alignment radius.

Cohesion, as a behavioural rule, directs each agent (or boid) to move towards the average position of its nearby flockmates. Figure 2.8c shows the mechanism of cohesion, with the bird in orange moving towards the centre c_c of its neighbours. This rule is crucial for the flock to stay together as a cohesive unit, especially during movement across expansive spaces or around obstacles. It helps to prevent the group from splintering and maintains the overall structural formation of the flock. Assume that there are N_c birds in the cohesion radius R_c of

bird j, the centre of mass within this radius can be found as

$$(x_{c_c}, y_{c_c}) = \frac{1}{N_c} \sum_{i}^{N_c} (x_i, y_i)\,, \tag{2.13}$$

where (x_i, y_i) represents the position of the i-th bird in the cohesion radius. The mathematical formula for cohesion update can then be described as follows:

$$a_{coh} = (a_{x_j}, a_{y_j}) = (x_c, y_c) - (x_j, y_j). \tag{2.14}$$

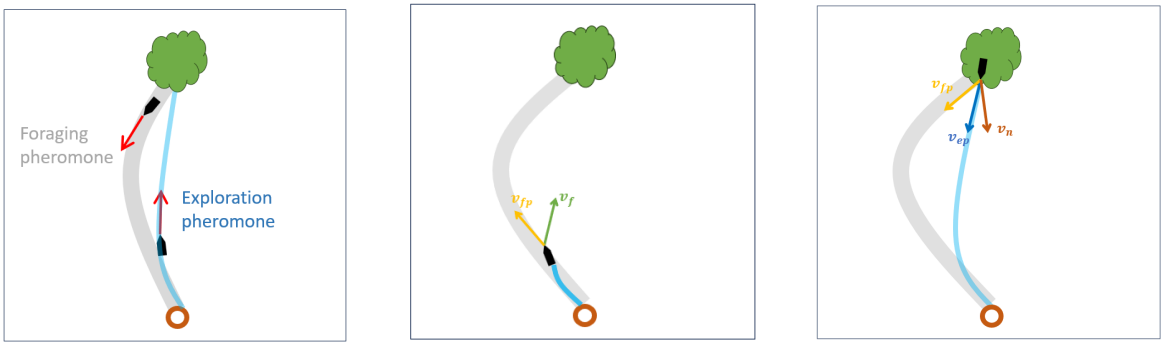
The update formula for cohesion seems the opposite of the one with separation, but they will not cancel each other out as they have different effective ranges ($N_s \neq N_c$). The velocity at time t can then be updated such that

$$v_t = v_{t-1} + a_{\text{total}}. \tag{2.15}$$

If the amount of velocity is greater than the maximum amount allowed, its magnitude will be rescaled to v_{max}:

$$(v_{x_j}, v_{y_j}) = \frac{(v_{x_j}, v_{y_j})}{\left\|(v_{x_j}, v_{y_j})\right\|} \times v_{max}. \tag{2.16}$$

2.4.2 Ants Model



(a) (b) (c)

Fig. 2.9 (a) Different pheromones used by ants, with exploration pheromone (blue) used when searching for food, and foraging pheromone (grey) used when carrying food back to the nest. (b) The velocity of an ant at the exploration phase is affected by both the foraging pheromone and the direction of food, resulting the overall velocity of the ant being a combination of v_f and v_{fp}. (c) The velocity of an ant at the foraging phase is affected by the foraging pheromone, exploration pheromone and direction of food, resulting in the overall velocity of the ant to be a combination of v_f, v_{ep} and v_{fp}.

Studies on ant behaviours have shown that ants utilise multiple types of pheromones (Dussutour et al., 2009). Each ant releases exploration pheromones when travelling towards food, while releasing forage pheromones when carrying food back to the nest. This is illustrated in Figure 2.9a. The exploration pheromones last longer but only have significant effects on the ant that released them, while foraging pheromones last shorter but have influences on the behaviour of all the ants.

When reaching food, the food is assumed to be visible to all ants. In the meantime, ants are also able to sense forage pheromones. Researches on ants behaviours have shown that both food position and pheromone trails would influence the route of ants, especially when food changes position (Beckers et al., 1990; Grüter et al., 2011). The direction the ants travel is determined based on the direction of food and pheromone trails, both acting as a velocity component for velocity update. For each individual ant, the velocity component v_{f} for the direction of food is calculated by the difference between the food position (x_f, y_f) and ant position (x, y):

$$v_f = (v_{f_x}, v_{f_y}) = \frac{(x_f, y_f) - (x, y)}{\left\|(x_f, y_f) - (x, y)\right\|} \tag{2.17}$$

The ants can sense pheromones in 3 directions: front, left and right. Sensing forage pheromones in each direction would result in a different velocity component:

$$v_{fp} = \begin{cases} (1,0) & \text{for front} \\ (0.33,-0.67) & \text{for left} \\ (0.33,0.67) & \text{for right} \end{cases} \tag{2.18}$$

Therefore the total update in velocity would be the weighted sum of the velocity components, with normalisation applied to limit the speed:

$$v_{total} = \frac{k_f v_f + k_{fp} v_{fp}}{\left\|k_f v_f + k_{fp} v_{fp}\right\|} \tag{2.19}$$

where k_f and k_{fp} are weighting factors for velocity components of foods and foraging pheromones. The effect of combining velocity components is shown in Figure 2.9b. Each individual ant has different responses when encountering pheromones (Isaeva, 2012). Some may more more likely to follow the pheromone trails and some may be more likely to head in the general direction of their destination. Therefore, every ant in the system would have differences in values of weights k_f and k_{fp}.

After the ants have collected the food, the travel direction will be decided based on the direction of the nest, the pheromone trail of itself while foraging or the pheromone trails of

other ants on their way back. The velocity component for nest v_n is related by the difference between the position of the nest (x_n, y_n) and food, calculated by:

$$v_n = (v_{n_x}, v_{n_y}) = \frac{(x,y) - (x_n, y_n)}{\|(x,y) - (x_n, y_n)\|}. \tag{2.20}$$

The velocity components v_{fp} and v_{ep} for both types of pheromones (foraging and exploration) are the same as in equation (2.18). Similar to the exploration phase, the total update in velocity is also the weighted sum of the velocity components, with normalisation applied to limit the speed:

$$v_{total} = \frac{k_n v_n + k_{fp} v_{fp} + k_{ep} v_{ep}}{\left\| k_n v_n + k_{fp} v_{fp} + k_{ep} v_{ep} \right\|} \tag{2.21}$$

where k_n, k_{ep} and k_{fp} are weighting factors for velocity components of foods and different pheromones. The effect of combining velocity components is shown in Figure 2.9c. As before, each individual ant has different responses when encountering pheromones, so every ant in the system would have differences in values of weights k_n, k_{fp} and k_{ep}.

2.5 Chapter Summary

Conclusively, this chapter examines the relevant literature in the field of reservoir computing (RC), specifically focusing on the use of physical robots and swarms for computational purposes. Swarm reservoirs offer scalability and flexibility advantages over single robot reservoirs. However, we discovered that existing research on using swarm reservoirs for reservoir computing (RC) is scarce, and there has been no exploration into the application of highly mobile, self-organising swarm systems as reservoirs. Thus, the subsequent chapter will detail our approach to filling this research gap.

Chapter 3

Methodology

This chapter will describe our reservoir computing framework, shown in Figure 3.1. In Section 3.1, we will first look at how the swarm models can be modified as reservoirs that exhibit echo state property described previously in Section2.1. Subsequently, Section 3.2 describes the permutation symmetry issue in our swarm reservoirs and we approach to handle it. Then we will illustrate how the readout layer is trained to map the reservoir states to the desired output in Section 3.3. Lastly, Section 3.4 summarises this whole chapter describing our framework.

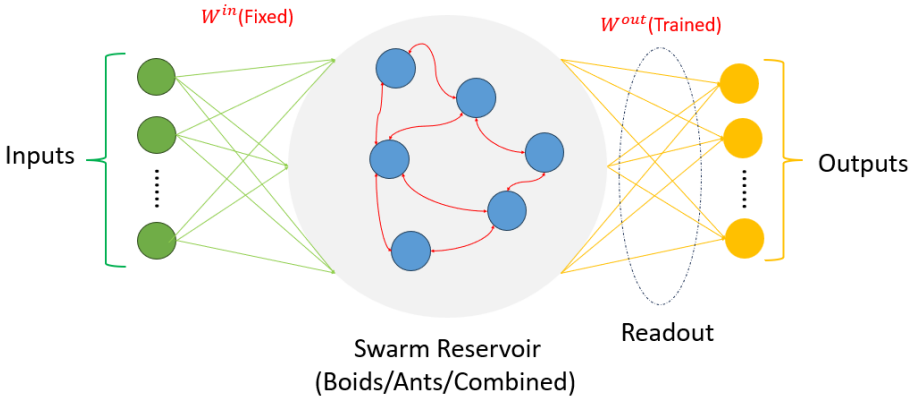


Fig. 3.1 The modified reservoir computing framework that we use, with the addition of an observation layer to overcome permutation symmetry

3.1 Swarm Reservoir Models

This section describes how swarm models mentioned in Section 2.4 are modified and applied in reservoir computers to perform machine learning tasks. Both models exhibit complex nonlinear dynamics which provides nonlinearity capacity for computation. With the modi-

fications applied, their outputs will also exhibit echo state properties, which offer memory capacities in computation.

3.1.1 Boids as Reservoir

If we consider each bird as a reservoir node, the edges would then be the interaction among them. Since all the birds only interact with their nearby flockmates, together they form a sparsely connected reservoir network. As birds in this system bird are all highly mobile, the connection of the reservoir also changes as the swarm proceeds.

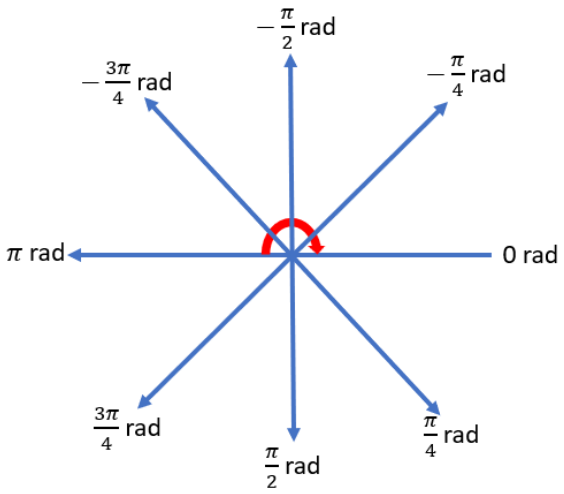


Fig. 3.2 Angle map of the birds.

The angle map of the birds is shown in Figure 3.2. In the boids model, all the birds are initially set at rest, randomly distributed within a certain range with all steering angles equal to 0. A bird would change its steering angle when the nearby birds turn, as it needs to align with others. As shown in Figure 3.3a, if Birds 1 and 2 make a clockwise turn, their nearby flockmates will then follow, aligning with them as shown in Figure 3.3b and 3.3c. Studies have suggested that for a flock with a larger amount of birds, this progress would propagate among the group of birds, creating a variety of delays with input (Potts, 1984; Reynolds, 1987). When some of the birds in the flock change the direction of flight, a 'manoeuvre wave' would propagate among the flock, with birds at the back gradually turning to the direction of birds in the front. In other words, the birds closer to the front would turn with a shorter delay, while the birds further to the front would turn at a longer delay.

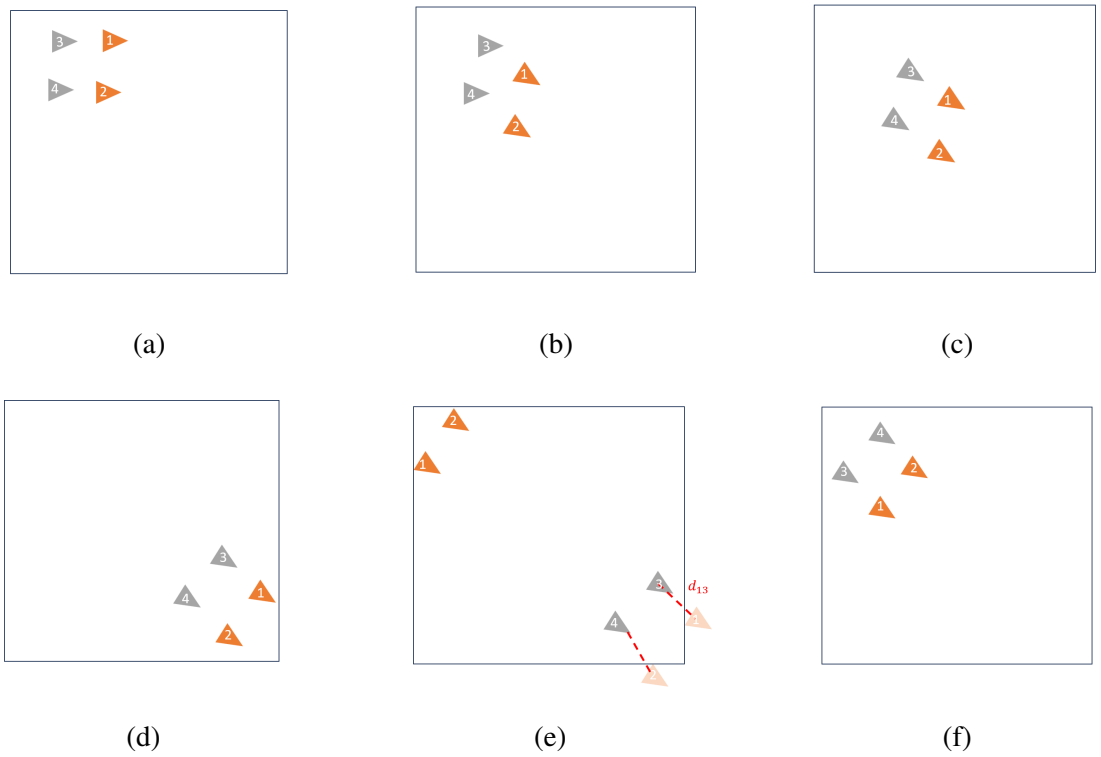


Fig. 3.3 (a) Birds initially at rest with steering angle at 0. (b) When Birds 1 and 2 change their steering angle. (c) Birds 3 and 4 steer to follow and align with the birds in the front. (d) When birds get close to the boundary. (e) Handling situations of birds reaching boundary, by making them reappear at the other side. (f) Birds reappear at the other side of the screen, and are still flying in a group

If some of the leading birds are compelled to steer based on input, the steering angles of the birds would demonstrate echo state properties, since the remaining birds would align and react to the input with varying delays. This makes the steering angle a suitable choice as both input and output for the reservoir.

Input is applied to some of the birds, and those birds with the input applied still follow the rules of separation, alignment and cohesion. The input acts as an additional element of force, and the total force acting on an input bird becomes:

$$F_{total} = k_{sep}F_{sep} + k_{ali}F_{ali} + k_{coh}F_{coh} + 2F_{in} \tag{3.1}$$

The update formula for acceleration is also changed accordingly:

$$a_{total} = k_{sep}a_{sep} + k_{ali}a_{ali} + k_{coh}a_{coh} + 2a_{in} \tag{3.2}$$

The states S_{boids} of the boids system are computed by concatenating angles θ_{bn} of each individual bird:

$$S_{boids} = [\theta_{b1}, \theta_{b2}, \ldots, \theta_{bN}] \tag{3.3}$$

The sky in reality is an unlimited space for birds to travel freely. However, in most simulations, there will be a boundary for the simulated 'sky', where the birds would turn around after reaching that. This brings unexpected distortions to the model dynamics. To solve this issue, the boundary handling mechanism was changed such that when a bird passed the boundary, it would appear at the other end of the screen. Denoting screen width as w_{screen} and screen height as h_{screen}. In Figure 3.3d, when Bird 1 reaches the right boundary with coordinate (w_{screen}, y_1), it would reappear at the left side of the screen with coordinate $(0, h_{screen} - y_1)$, shown in Figure 3.3e. In this way, all 4 birds will still stick together after passing the boundary, as shown in Figure 3.3f. The positions of birds 1 and 3 are flipped with birds 2 and 4, but this flip would not affect their steering angles. With the merge of boundaries, the way that distance between birds is calculated also needs to be changed. In Figure 3.3e, the distance d_{13} between birds 1 and 3 can be calculated as

$$d_{13} = \|(x_1, y_1) - (x_3, y_3)\| \tag{3.4}$$

Then when bird 1 crosses the boundary and reaches the other side, the way to calculate distance changes to

$$d_{13} = \|(x_1, -y_1) + (w_{screen}, h_{screen}) - (x_3, y_3)\| \tag{3.5}$$

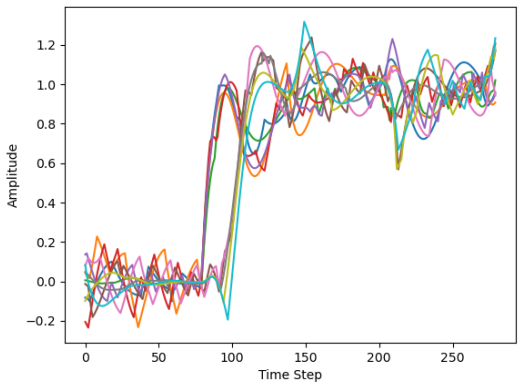
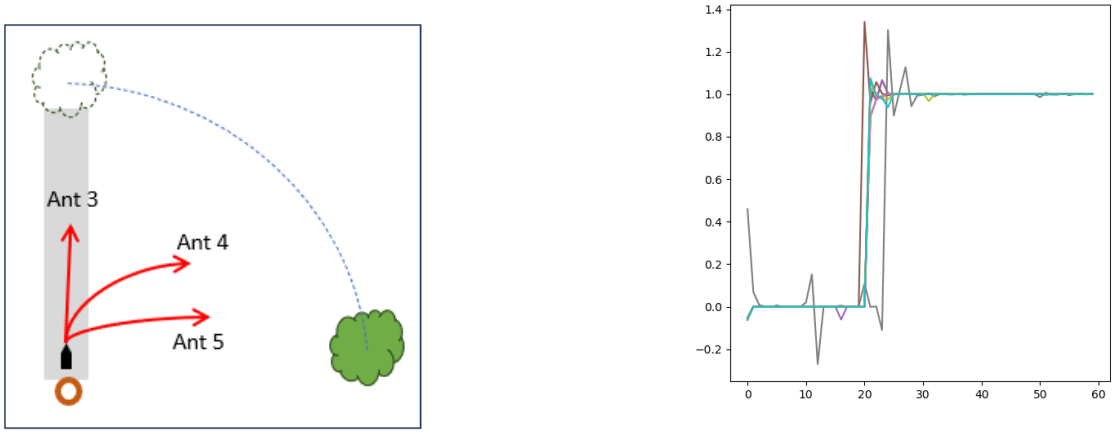


Fig. 3.4 Illustration of echo state property in our boids reservoir under a step input, with different birds exhibiting different amount of delays to input.

Figure 3.4 illustrates the states of birds when a step input is applied at time step 80. It can be seen that the birds respond to the input at different delays, verifying the echo state property of the reservoir. Note that the fluctuation is because the birds are highly mobile across space. The echo state property makes the boids reservoir exhibit memory capacity, while the nonlinearity capacity comes from the complex nonlinear dynamics within the system.

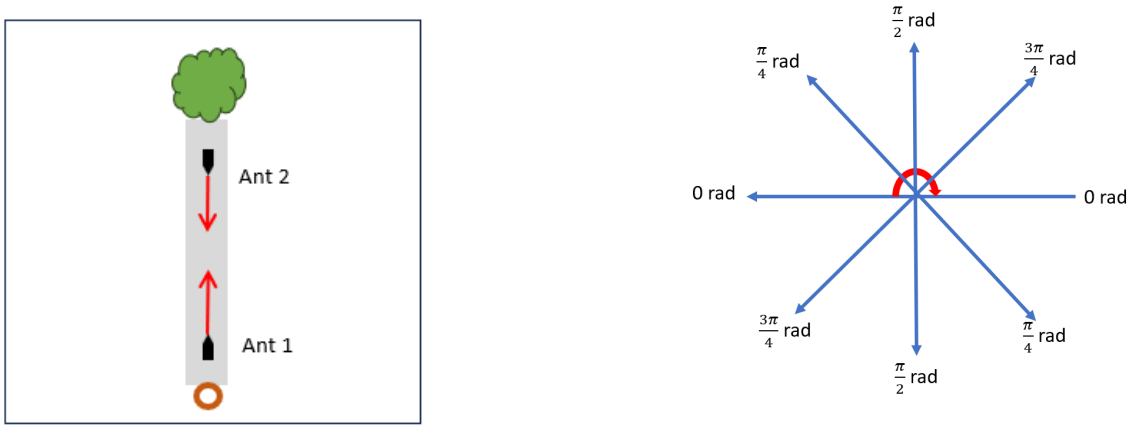
3.1.2 Ants as Reservoir



(a) (b)

Fig. 3.5 (a) Different ants respond to changes in food source differently, resulting in differences in delays. (b) Illustration of echo state property in our ants reservoir under a step input, with different ants exhibiting different amounts of delays to input.

In the ant reservoir, the input is chosen as the angle of food relative to the nest, and the output is chosen as the steering angle of ants. In every time step, a new pile of food will be placed at an angle that corresponds to the input, and the food at the last time step will be removed. The food at different time steps is kept at the same distance relative to the nest, therefore they only vary in direction relative to the nest. When the position of food changes, ants may still follow the pheromone trails in the previous directions. This creates a delay between input and reservoir states. The delays between different ants within the reservoir come from the different factors k_f and k_{fp} for velocity components for different ants in Equation 2.19 and 2.21. The effect is illustrated in Figure 3.5a. This forms echo state property, which is shown in Figure 3.5b. The echo state property makes the ants reservoir exhibit memory capacity, while the nonlinearity capacity comes from the complex dynamics within the system.



(a) (b)

Fig. 3.6 (a) Ants travelling in opposite directions in the exploring and foraging phase, leading to inconsistency in steering angles. (b) Angle map of ants modified to solve this inconsistency.

However, the ants may travel back and forth between the nest and the food. For example, in Figure 3.6a, ant 1 is travelling towards the food with a steering angle measured as $\frac{\pi}{2}$ rad. Meanwhile, ant 2 is travelling in the opposite direction towards the nest, with the steering angle measured as $\frac{3\pi}{2}$ rad. To solve this inconsistency, all angle values above π rad are mapped to their opposite directions by subtracting π. The resultant angle map for ants is shown in Figure 3.6b. The states S_{ants} of the ants system are computed by concatenating the angles of each individual ant:

$$S_{ants} = [\theta_{a1}, \theta_{a2}, \ldots, \theta_{aN}]. \tag{3.6}$$

3.1.3 Combined Reservoir

Studies have proven that when the reservoir contains more dynamics, the reservoir computer can harness the advantages from different dynamics and perform computation tasks in a better way. The study from Holzmann and Hauser (2010) shows the effectiveness of adding artificial delays to ESN. Their modification brings improvements to the memory capacities of ESN, resulting in a better performance in the Markey-Glass time series prediction task. Inspired by their work, we combine the boids and ants reservoirs to investigate if harnessing the dynamics of different swarm systems can bring benefits for computation. We assume that there is no interaction between the two swarm reservoirs. Therefore, the combination is achieved by running the two swarm reservoirs separately in parallel, and then merging their

states through concatenation. The states S_c of the combined reservoir are computed by

$$S_c = [S_{ants}, S_{boids}]. \tag{3.7}$$

3.2 Observation Layer

This section first introduces the issue of permutation symmetry in our swarm reservoirs, then explains details of the observation layer that we apply to address permutation symmetry. This observation layer is applied to process the output of the reservoir before readout. This layer computes outputs based on the distribution of reservoir states, ensuring the resulting output remains invariant to internal permutations within the reservoir.

3.2.1 Permutation Symmetry in Our Swarm Reservoirs

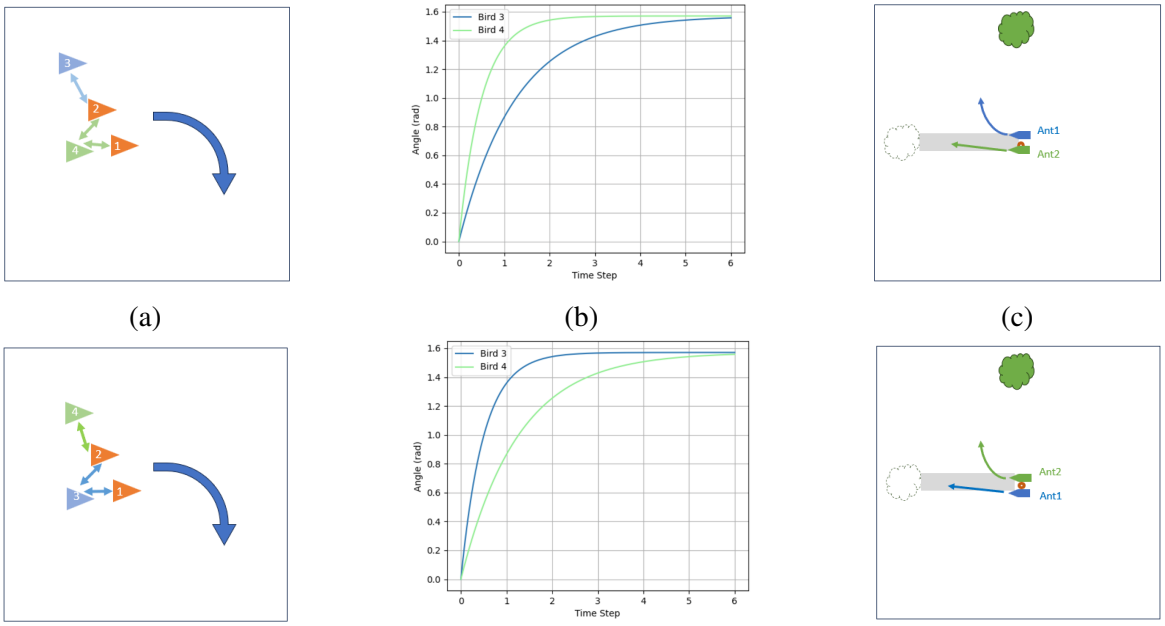


Fig. 3.7 Permutation symmetry in both reservoirs. (a) A flock of 4 birds, with Birds 1 and 2 receiving input, changing direction of flight. (b) Variation in steering angle of Birds 3 and 4. (c) Behaviour of different ants when food source changes, with Ant 3 turning to food and Ant 4 remain travelling along pheromone trail. (d) A flock of 4 birds with Birds 3 and 4 switched position. (e) Variation in steering angle of Birds 3 and 4 when position is switched. (f) When Ants 3 and 4 switched position compared to (c), resulting Ant 4 turning to food and Ant 3 remain travelling along pheromone trail.

The paper by Lymburn et al. (2021) discussed that swarm reservoirs generally exhibit permutation symmetry, which degrades the performance of the reservoir. The issue is generally caused by the agents exchanging positions with each other as they proceed. The boids and ants model in our case also exhibits permutation symmetry. As shown in Figure 3.7a, considering a swarm group of four boid agents, with Birds 1 and 2 receiving inputs. When both birds receive an input of $\pi/2$, they will perform a clockwise rotation of this angle. Then birds 3 and 4 will also respond accordingly. It can be seen that both Bird 1 and 2 are in the alignment range of Bird 4, while Bird 3 only has Bird 2 in its alignment range. From equation (2.12), it can be seen that having a larger amount of birds in the alignment range would have a larger acceleration component. This means that Bird 4 will respond faster to the changes in angles of the nearby birds, compared to Bird 3. The variation in angles with time for the two birds is shown in Figure 3.7b. When Birds 3 and 4 swap position during flight, as shown in Figure 3.7d , Bird 3 now has two other birds in its alignment range and the number of birds in the alignment range of Bird 4 becomes one. Bird 3 now responds faster to the input than Bird 4. This inconsistency in delay amount for the same node would make it very hard to train the readout layer.

The ants model exhibits a similar situation as well. The pheromone trail left by a group of ants will accumulate to form a thicker trail. Then ants might appear at different positions within this trail. As shown in Figure 3.7c, when the angle of food relative to the nest changes from 0 to $\pi/2$, Ant 3 is travelling on the side of the pheromone trail that was closer to the food compared to Ant 4. Equation (2.19) suggests that the direction of food would act as a velocity component, making the steering angle of ants deviate to the new food position. Due to the difference in position, Ant 3 will leave the pheromone trail sooner than Ant 4. When Ant 3 leaves the pheromone trail, the velocity component for pheromones disappears. Then Ant 3 will head directly to the direction of food giving a state output close to $\pi/2$, while the state of Ant 4 remains at approximately 0. When Ant 3 and 4 switch positions, as shown in Figure 3.7f, Ant 4 now leaves the pheromone trail before Ant 3. Then the state of Ant 3 and 4 will be the opposite of the one in Figure 3.7c, under the same input condition.

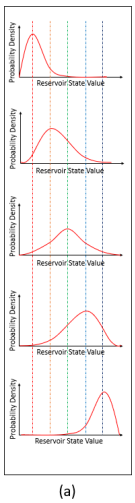
3.2.2 Addressing Permutation Symmetry

To solve permutation symmetry, an additional observation layer is applied. This symmetry originates from agents switching positions with one another, which means the overall distribution of the states would remain the same. Therefore, we apply an observation layer between the reservoir and outputs to consider the overall distribution only. This distribution is obtained by computing the probability density function for all states. The probability density function (PDF) is calculated by applying Gaussian kernel functions to fit all data

points. This approach involves using each data point as the centre of a Gaussian kernel, with the sum of these kernels providing a smooth estimate of the density distribution. The kernels have a predefined bandwidth, which determines their width and influences the smoothness of the resulting density function. This bandwidth parameter is crucial as it balances the trade-off between the bias and variance of the density estimate, affecting the accuracy and generalisability of the model. In our case, the 'Silverman's rule of thumb' (Silverman, 2018) is first applied to automatically determine the bandwidth.

$$h = \left(\frac{4\hat{\sigma}^5}{3n} \right)^{\frac{1}{5}}, \tag{3.8}$$

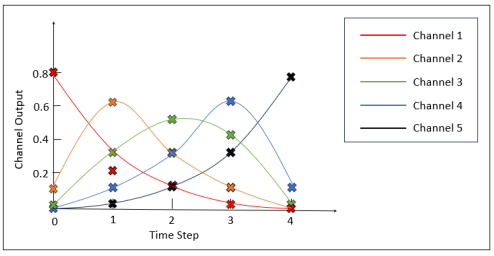
where h is the bandwidth, $\hat{\sigma}$ is the estimated standard deviation of the data, and n is the number of data points. Then the resulting bandwidth is divided by a factor of 2 to make the kernel density estimate more sensitive to the data. By employing Gaussian kernels, this method effectively captures the underlying trends within the data, providing a continuous, smooth approximation of distribution. We then observe the range of state values that this PDF generally covers, and uniformly take 40 points within this range. For the benchmark tasks that we implemented, the range is generally between -0.5 rad and 1.5 rad. Consequently, the output of the observation layer is obtained by taking the probability density values corresponding to the 40 channels we choose.



(a)

Channel	Time step				
	0	1	2	3	4
1	0.8	0.2	0.1	0.01	0
2	0.1	0.6	0.3	0.1	0
3	0.01	0.3	0.5	0.4	0.01
4	0	0.1	0.3	0.6	0.1
5	0	0.01	0.1	0.3	0.8

(b)



(c)

Fig. 3.8 Working principle of the observation layer. (a) Changes in distribution of reservoir states when input is changed from 0 to 1, across 5 time steps. (b) Probability density outputs from the 5 channels over 5 timesteps. (c) Plot of observation channel output against time step, exhibiting echo state property.

The resulting output would be a transformation of the state values, with variable amount of delays that ensure computation capacity. Figure 3.8 shows an example of how an observation layer with 5 channels operates. When the angles (states) of the swarm system gradually change from 0 rad to 1 rad, the original left-skewed PDF will skew towards 1, shown in Figure 3.8a. Then Figure 3.8b shows the process of taking the probability density values of each channel at different time steps. It can be seen that the resulting output in Figure 3.8c exhibits the delay pattern that Section 2.1 describes, as the output peaks at different time steps.

3.3 Readout Layer

This section first introduces the readout layer, which maps output from our observation layer to the target output. Then output evaluation method through covariance is discussed. However, note that reservoir computing employs a one-time training process without iterations. Therefore, the output evaluation is only for performance analysis by us, not for training. The training and evaluation approach is similar to the one of ESNs (Jaeger, 2001; Jaeger and Haas, 2004).

3.3.1 Training readout weights

After obtaining the reservoir states, the readout weights $\mathbf{W}_{\text{out}}$ are needed to be trained to map the reservoir states to the desired output. For a reservoir of N nodes with an input sequence of length n applied, the output matrix X for a given time step is constructed by concatenating the bias vector b, the input vector x, and the reservoir states, resulting $\mathbf{X} \in \mathbb{R}^{n \times N+2}$. Mathematically, it can be expressed as:

$$\mathbf{X} = [b, x, \mathrm{S}] \tag{3.9}$$

where $b \in \mathbb{R}^{n \times 1}$ is the bias term that adds a constant offset to the transformation, ensuring non-linearity and helping to avoid nodes being stuck in undesirable states (like zero states). $x \in \mathbb{R}^{n \times 1}$ is the input applied to the reservoir, $\mathrm{S} \in \mathbb{R}^{n \times N}$ contains the memory of the past states of all reservoir nodes in the past n time steps. The initial outputs of the reservoir are often not representative of the steady-state behaviour the network will exhibit once fully driven by the input. The washout period allows the reservoir to overcome these initial transient conditions, ensuring that the states are fully reflective of the input signal dynamics. The washout of m time steps is applied to the output matrix such that the first m rows are removed and not included in readout training, resulting $\mathbf{X}_{\mathbf{washed}} \in \mathbb{R}^{(n-m) \times N+2}$. Ridge regression is used to

compute the readout weights by finding a linear combination of reservoir states that best approximate the desired output:

$$\mathbf{W}_{\text{out}} = \mathbf{Y}^{\top}\mathbf{X}_{\mathbf{washed}}\left(\mathbf{X}_{\mathbf{washed}}{}^{\top}\mathbf{X}_{\mathbf{washed}} + \lambda\mathbf{I}\right)^{-1} \tag{3.10}$$

where $\mathbf{X}_{\mathbf{washed}} \in \mathbb{R}^{(n-m)\times N+2}$ is the output matrix mentioned previously, $\mathbf{Y} \in \mathbb{R}^{n\times 1}$ is target output matrix corresponding to the training inputs, λ is the regularisation parameter that controls the trade-off between fitting the training data well and keeping the weights small to avoid overfitting. We choose the value of λ as 1×10^{-4} to balance this trade-off.

3.3.2 Output Computation and Evaluation

The training output $\mathbf{O}_t \in \mathbb{R}^{n\times 1}$ is computed using:

$$\mathbf{O}_t = (\mathbf{W}_{\text{out}}\mathbf{X}^T)^T. \tag{3.11}$$

The performance is evaluated by comparison with target training output $\mathbf{T}_t \in \mathbb{R}^{n\times 1}$, through covariance which measures the similarity between $\mathbf{O}_t$ and $\mathbf{T}_t$. The covariance matrix is formed by:

$$\text{Cov}(\mathbf{O}_t, \mathbf{T}_t) = \begin{bmatrix} \sigma_{\mathbf{O}_t\mathbf{O}_t} & \sigma_{\mathbf{O}_t\mathbf{T}_t} \\ \sigma_{\mathbf{T}_t\mathbf{O}_t} & \sigma_{\mathbf{T}_t\mathbf{T}_t} \end{bmatrix} \tag{3.12}$$

where $\sigma_{\mathbf{O}_t\mathbf{O}_t}$ is the variance of $\mathbf{O}_t$, $\sigma_{\mathbf{T}_t\mathbf{T}_t}$ is the variance of $\mathbf{T}_t$, $\sigma_{\mathbf{O}_t\mathbf{T}_t} = \sigma_{\mathbf{T}_t\mathbf{O}_t}$ is the covariance between $\mathbf{O}_t$ and $\mathbf{T}_t$. Each element is calculated as:

$$\sigma_{\mathbf{O}_t\mathbf{O}_t} = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{O}_{t_i} - \bar{\mathbf{O}}_t)(\mathbf{O}_{t_i} - \bar{\mathbf{O}}_t) \tag{3.13}$$

$$\sigma_{\mathbf{T}_t\mathbf{T}_t} = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{T}_{t_i} - \bar{\mathbf{T}}_t)(\mathbf{T}_{t_i} - \bar{\mathbf{T}}_t) \tag{3.14}$$

$$\sigma_{\mathbf{O}_t\mathbf{T}_t} = \sigma_{\mathbf{T}_t\mathbf{O}_t} = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{O}_{t_i} - \bar{\mathbf{O}}_t)(\mathbf{T}_{t_i} - \bar{\mathbf{T}}_t) \tag{3.15}$$

where $\bar{\mathbf{O}}_t$ and $\bar{\mathbf{T}}_t$ are the mean of $\mathbf{O}_t$ and $\mathbf{T}_t$, and i enumerates all elements in the output. The training covariance can then be calculated as

$$\text{cov}_t = \frac{\sigma_{\mathbf{O}_t\mathbf{T}_t} \times \sigma_{\mathbf{T}_t\mathbf{O}_t}}{\sigma_{\mathbf{O}_t\mathbf{O}_t} \times \sigma_{\mathbf{T}_t\mathbf{T}_t}} \tag{3.16}$$

In the prediction phase, this process is repeated, with the predicted output $\mathbf{O}_p$ computed using the trained readout weights $\mathbf{W}_{\text{out}}$. The predicted output is then evaluated by computing covariance with target output $\mathbf{T}_p$, using the same approach specified from Equation 3.12 to Equation 3.16

3.4 Chapter Summary

To summarise, this chapter describes our reservoir computing framework. We modified the swarm models described in Section 2.4 into reservoirs that exhibit echo state property. The issue of permutation symmetry is also described and addressed through Gaussian kernel density estimation. The method for readout weights training is also discussed. In the next chapter, we will examine the computation capacity of our approach on the benchmark datasets, and compare the performance with the classical RC framework.

Chapter 4

Experimental Setups and Results

In this chapter, the performance of our reservoir computing system on the benchmark tasks will be shown. Section 4.1 introduces the benchmark tasks, then Section 4.2 shows the experiment settings including how the parameters are tuned and set. Section 4.3 shows our most significant result, with the performance of our swarm reservoirs compared to classical echo state networks (ESNs) described in Section 2.1. The following few sections will include a detailed analysis of variations in computation performance with model settings. Section 4.4 shows changes in computation capacities as we increase the size of the swarm used in the reservoir. The effectiveness of our solution for permutation symmetry will also be illustrated. Following this, Section 4.5 demonstrates the changes in computation capacity when the two types of swarm reservoirs are combined in parallel at different ratios. Furthermore, Section 4.6 looks into the impact of modifying some of the model parameters on performance. Finally, Section 4.7 briefly concludes about our results and findings.

4.1 Benchmark Tasks

This section introduces the four benchmark tasks for reservoir computing that we use to evaluate computation capacity, which come from the paper by Jaeger and Haas (2004). The predicted outputs and target outputs are also visually compared.

4.1.1 Short-Term Memory Task

The short-term memory (STM) task measures the reservoir's ability to recall a sequence of bits or continuous values over short periods. This task is designed to simulate a function that produces a version of the input stream delayed by a specified number of time steps. Therefore, performing this task only requires memory capacity.

This task contains 10 subtasks, from STM1 to STM10. Here STMx means that the reservoir computing framework needs to reproduce the input stream delayed by x steps. Performance on this task is typically evaluated by how accurately the system can reproduce a sequence of inputs after varying delay intervals.

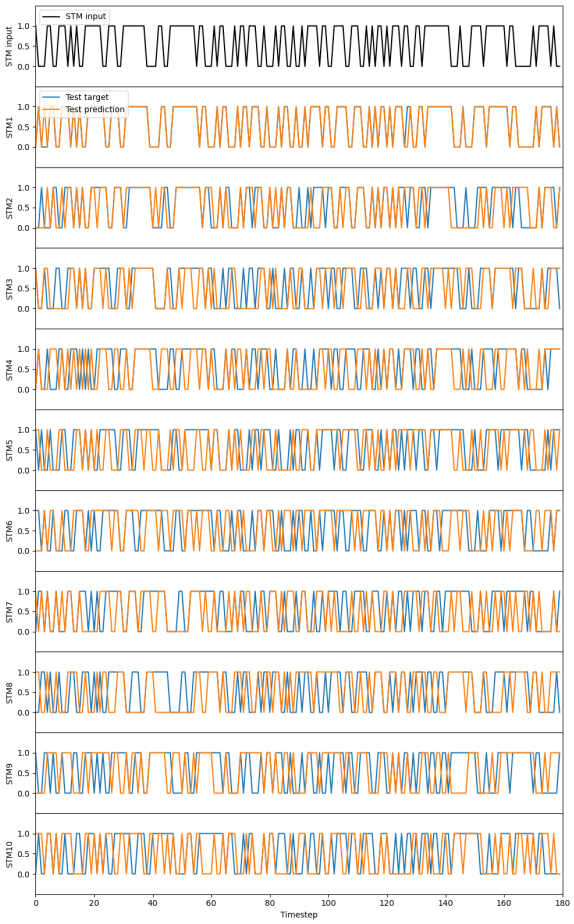


Fig. 4.1 Performance of our combined reservoir in STM tasks. Longer memories are required for subtasks with larger indices. Our reservoir can fully perform STM1 and partially perform STM2 and STM3, meaning that it only has memories for the previous 3 inputs.

The performance of our combined reservoir in STM tasks is shown in Fig. 4.1. As the required delay between the input and output increases, it can be seen that the performance gradually degrades. The desired target for STM1 subtask can be produced with a high accuracy, while the target for STM2 and STM3 can only be partially produced. Then after STM4, only a very small portion of the target output can be correctly generated. The result shows that our reservoir has having memory of the past three inputs.

4.1.2 Parity Check Task

The Parity Check (PC) task involves determining the parity (odd or even nature) of a sequence of bits. It involves emulating a nonlinear function that checks the parity of a sequence of binary inputs. Within a certain delay period, if the sum of input is even, the parity checker outputs 0, indicating even parity. Otherwise, it outputs 1, indicating odd parity. Performing this task requires both memory and nonlinearity capacity.

This task contains 10 subtasks, from PC1 to PC10. The subtask's index indicates the amount of delay that the parity check operates on.

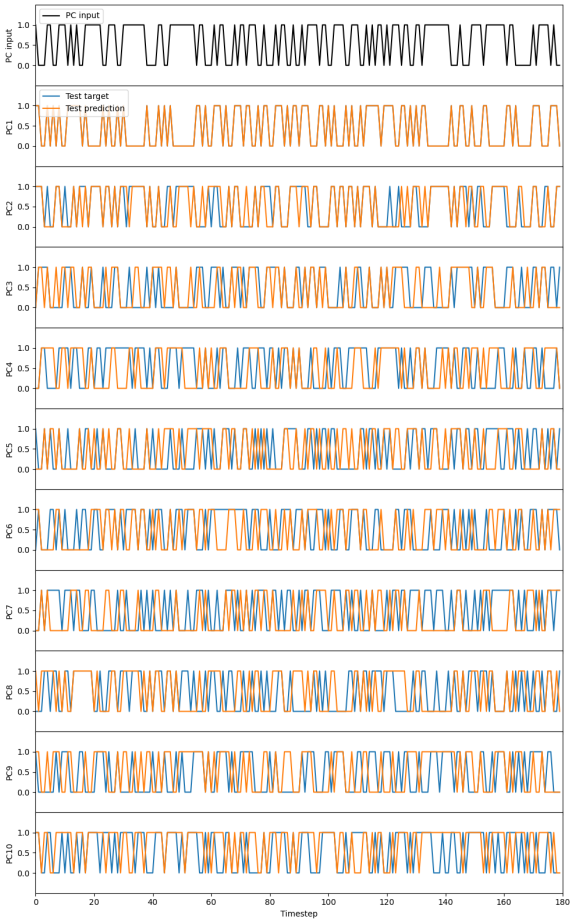


Fig. 4.2 Performance of our combined reservoir in PC tasks. Longer memories are required for subtasks with larger indices. Our reservoir can fully perform PC1 and partially perform PC2 and PC3, validating that it only has memories for the previous 3 inputs.

The performance of our combined reservoir in PC tasks is shown in Fig. 4.2. As the required delay between the input and output increases, it can be seen that the performance gradually degrades. The target output for the PC1 subtask can be generated with high

accuracy, whereas the targets for PC2 and PC3 are only partially achievable. Following PC4, only a minimal fraction of the target output can be accurately produced. The result shows that our reservoir can only perform parity checks for the past three inputs, which agrees with the performance of STM tasks that reservoir is having memory of the past three inputs.

4.1.3 Nonlinear Autoregressive Moving Average Task

The Nonlinear Autoregressive Moving Average (NARMA) task is designed to assess both the memory and the nonlinearity handling capabilities of a reservoir. This task involves predicting the next value in a nonlinear and time-dependent series, where each next step depends heavily on both the current input and a series of past inputs. This task is a representation of a reservoir computer emulating a dynamical system driven by inputs.

The NARMA task contains 5 subtasks, including NARMA2, NARMA5, NARMA10, NARMA15, NARMA20. The index of subtasks represents the order of nonlinear dynamical systems that the reservoir computer needs to emulate.

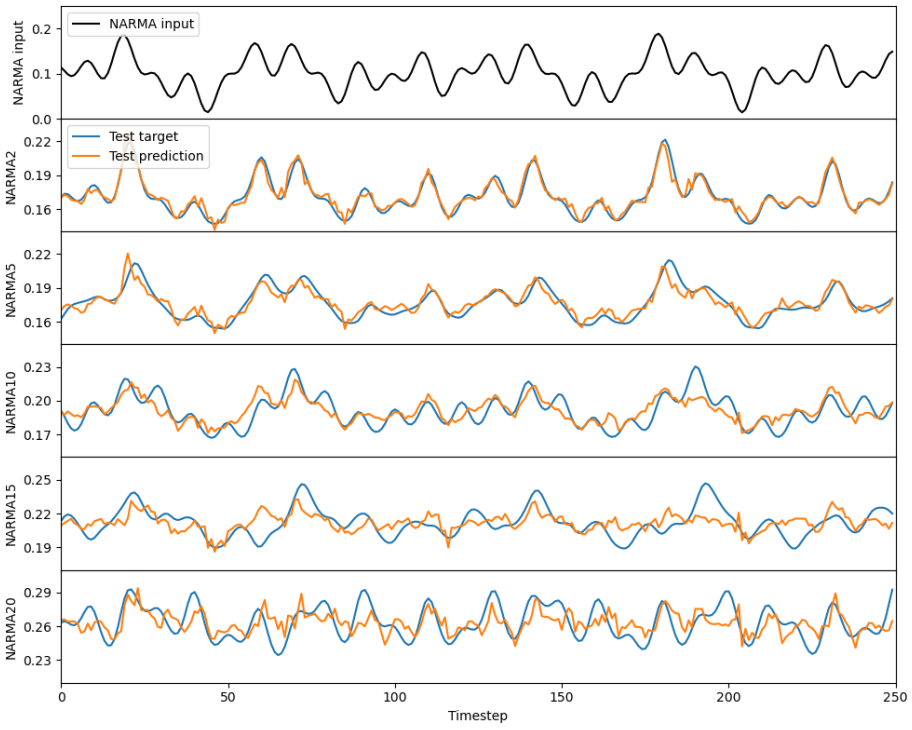


Fig. 4.3 Performance of our combined reservoir in NARMA tasks. Larger subtask indices represent larger order of nonlinearity. Our reservoir emulates nonlinear systems of lower order (NARMA2 and NARMA5) well, while non-linear systems of higher orders can only be partially emulated.

Fig. 4.3 shows the performance of our reservoir computer for NARMA tasks. The performance degrades as the order increases. Our reservoir computer performs a good emulation of the nonlinear dynamical system of order 2 and 5, with target output produced in a relatively accurate manner. Meanwhile, it can only partially achieve the target output of the nonlinear systems with higher orders.

4.1.4 Mackey-Glass Time Series Prediction Task

Similar to the NARMA task, the Mackey-Glass (MG) task is a time series prediction problem based on a delay differential equation that is known for its chaotic behaviour. However, this task requires the reservoir computer to emulate a dynamical system driven by itself. Instead of using a predefined input series, this task requires using past outputs as inputs to predict future values in a series, emphasising the system's memory and nonlinear dynamics processing capabilities.

The MG task contains 2 subtasks: MG16 and MG17. The 16 and 17 represent the delay that the MG system incorporates. The system displays a chaotic attractor when the delay exceeds 16.8. Therefore, the two subtasks represent both non-chaotic and chaotic scenarios.

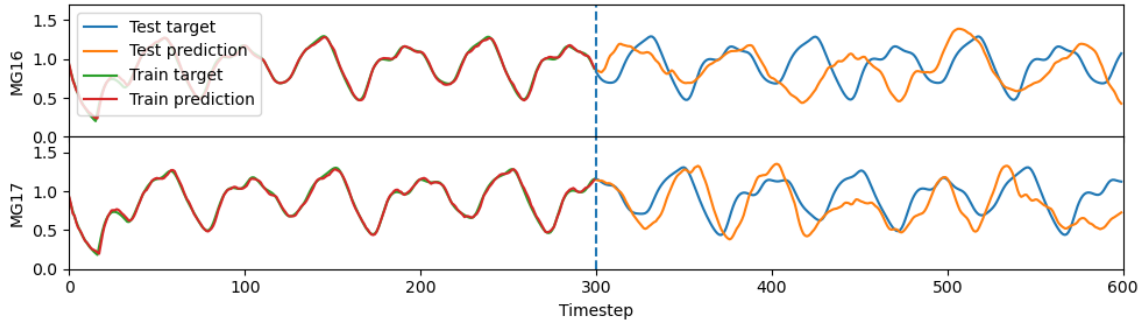


Fig. 4.4 Performance of our combined reservoir in MG tasks. MG16 represents a non-chaotic MG system and MG17 represents a chaotic MG system. The predicted output mostly fits the pattern of the target output.

This is a challenging task that most reservoirs do not perform well. Fig. 4.4 shows the performance of our reservoir computer for MG tasks. For both subtasks, the reservoir computer gives predictions that mostly fit the general pattern of the target.

Table 4.1 Parameters and Variation Ranges for Boids Simulation

Parameters	Standard Values	Variation Ranges
Separation factor k_{sep}	0.05	0–0.25
Separation range (unit length)	20	0–100
Alignment factor k_{ali}	0.05	0-0.25
Alignment range (unit length)	60	0–200
Cohesion factor k_{coh}	0.015	0-0.05
Cohesion range (unit length)	300	0–400
Input ratio	0.5	0–1
Delay for STM (time step)	16	10–20
Delay for PC (time step)	16	10–20
Delay for NARMA (time step)	4	1-10
Delay for MG (time step)	2	1-10

Table 4.2 Parameters and Variation Ranges for Ants Simulation

Parameters	Standard Values	Variation Ranges
Food factor k_f	$U(0.05,7)$	$U[0.05,u]$ for $u \in [0.1,10]$
Nest factor k_n	$U(0.05,7)$	$U[0.05,u]$ for $u \in [0.1,10]$
Exploration pheromone factor k_{ep}	1	-
Forage pheromone factor k_{fp}	1	-
Delay for STM (time step)	300	100–1000
Delay for PC (time step)	300	100–1000
Delay for NARMA (time step)	300	100–1000
Delay for MG (time step)	300	100–1000

4.2 Experimental Methods and Settings

The standard setting of parameters for the boids and ants reservoir are shown in Table 4.1 and Table 4.2 respectively.

Birds in nature exhibit a large cohesion radius, enabling widely separated flocks to reassemble (Reynolds, 1987). Therefore, our cohesion radius is set at a large value of 300. Separation mainly ensures birds do not collide with one another, therefore our separation radius is set at a small value of 20. As mentioned in Section 3.1.1, when the birds at the front change direction, birds align with their nearby flockmates, forming a 'manoeuvre wave ' among the flock. Studies have also suggested that birds can observe the incoming manoeuvre wave and begin synchronising their alignment before the wave arrives (Potts, 1984). In order to ensure the flock exhibits maneuver wave, while making the birds respond to it before its arrival, the alignment range is set as 60. The alignment, separation and cohesion factors are also set such that the bird's flock operates in a relatively stable manner, with the presence of

manoeuvre wave. The input ratio is set to 0.5, which is obtained from sensitivity analysis that will be discussed in Section 4.6. This is the minimum ratio for the boids reservoir to have desirable performance, where all birds follow the input.

For the ants model, the ratio between food/nest factor and pheromone factors varies a lot for different types of ants. Therefore, it is hard to get a numerical value from the existing studies. To ensure the diversity in behaviour for ant groups of larger sizes, we assume that the food and nest factor lies within a uniform distribution that ranges from 0.05 to 7. This choice is also validated by the sensitivity analysis later from Section 4.6.

The evaluation method for the study involves a systematic variation of parameters and testing across multiple benchmarks. We adjust one parameter at each time following Table 4.1 and 4.2, while keeping others at the standard values. Note that for the ant reservoir, Equation 2.19 and 2.21 suggest that total velocity is the sum of velocity components weighted by factors, with normalisation applied. Therefore, increasing the food/nest factor is equivalent to reducing pheromone factors and vice versa. Hence, only food/nest factors are varied in our sensitivity analysis.

The four benchmark tasks mentioned in the previous section, namely STM, PC, NARMA, and Mackey-Glass, are utilised to measure the system's capabilities. Our reservoirs need a certain amount of delay to respond to changes in input for each task. We choose the suitable amount delay for each task by testing reservoirs of different sizes, under a range of different delays. Appendix B shows an example of delay analysis for swarm reservoirs of size 40. The delay amount that achieves desirable performances for a variety size of swarms is then chosen, shown in Table 1. In most of the reservoir computing tasks, the state output of the reservoir is taken at the end of the delayed time, before the input changes. However, it is interesting to note that our ants reservoir would produce a better performance when state output is taken near the beginning of the delayed time. Therefore, for the ants model, we choose a delay amount of 300 time steps but take the reservoir states at an earlier time step. Section 3.3 mentions that washout is needed to be applied so that the readout training process only uses the results produced after the reservoir is stabilised. We observe the response of the reservoir and choose a washout period of 50 time steps for NARMA task. Then for STM and PC tasks, a washout period is chosen as 20 time steps, whereas for the MG task, no washout period is implemented.

The system's performance is assessed using different random seeds to ensure robustness and generalisability. For the boids reservoir, the random seed controls the initial position of the birds, while for the ants reservoir, the random seed determines how the food nest factors are sampled from the uniform distribution.

The capacity of the system is quantified through the covariance between the predicted output and the target output, providing a statistical measure of prediction accuracy. For each specific setup (swarm size, parameters, random seed, etc.), we compute the average covariance across all tasks, forming a data point for the plot. For example, in the study examining the effects of different swarm sizes in boids, we utilise 20 unique random seeds for each swarm size. This approach generates 20 covariance data points for each size. We then fit a polynomial to all data points, offering a comprehensive evaluation of the system's overall performance.

We also compare the performance of our reservoir computer with Echo State Netweorks (ESNs). For different network sizes, we test the performance of 500 random ESNs on our benchmark tasks, using identical settings to Lukoševičius (2012). We then take the average covariance as result for comparison.

4.3 Comparison with Echo State Networks

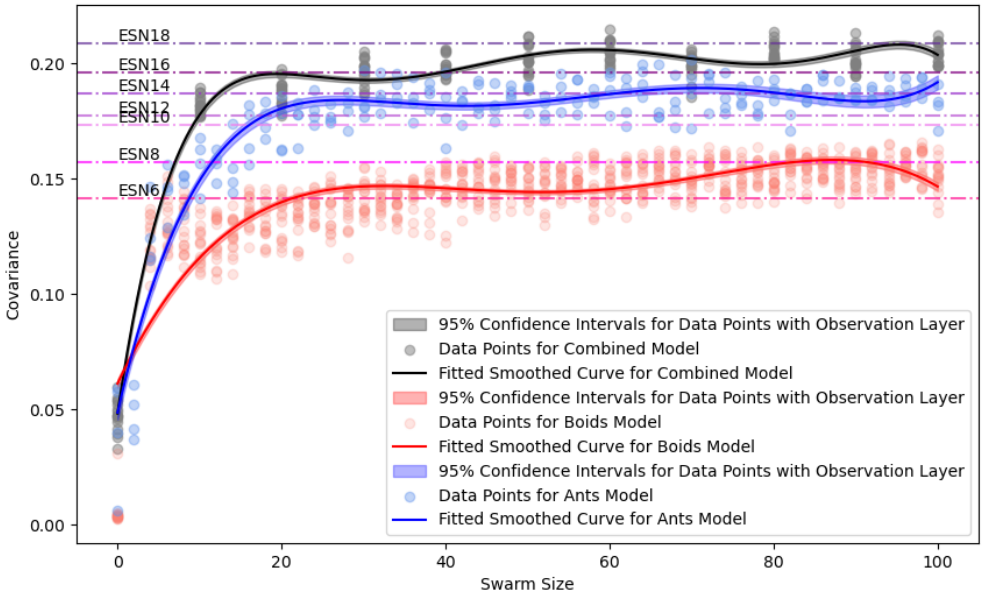


Fig. 4.5 Comparison of ESN with the three reservoirs of ours. At swarm size of 20, the performance for boids, ants and combined reservoirs are at levels of ESN6, ESN14 and ESN16, respectively. For larger swarm sizes, the performance for boids and combined reservoir approaches ESN8 and ESN18, while the capacity for ants reservoir remains at level of ESN14.

From Figure 4.5, it can be seen that when the swarm size reaches 20, the performance of our boids reservoir has reached a comparable level with ESN6. As for the ants and combined reservoir, the performance is equivalent to ESN14 and ESN16 respectively.

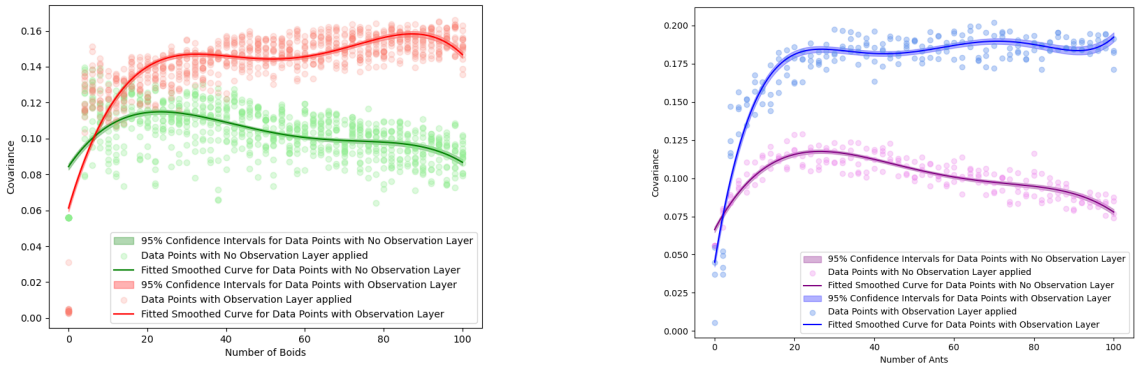
Nevertheless, our model still has a large gap with ESN of larger sizes, as the improvement in computation capacity becomes less significant when sizes get larger. When the size of the swarm gets above 60, the performance for boids and combined reservoir improved slightly to the level of ESN8 and ESN18, while the performance for ant reservoir remains at the level of ESN14. This undesirable performance is related to swarm dynamics, which exhibit limited memory capacity. Section 4.1.1 discussed that our swarm reservoirs can only partially perform the first 3 STM tasks, which means that our reservoirs only have memory for the past few states.

The limited memory capacity for the boids reservoir of larger sizes is caused by the previous states dying away too fast within the reservoir. Theoretically, the manoeuvre wave would travel longer within a larger swarm group, resulting in a greater variety of delays and better memory capacity. However, as the total number of swarms increases, the interactions between the swarm agents become more complicated, and the amount of interactions also varies more significantly with time. These interactions create forces that disturb the memory of past input states. The alignment force makes the input steering angle propagate among the flock, creating memories within the reservoir. Nevertheless, birds' steering angle is also affected by forces of separation and cohesion. Therefore, previous steering angle inputs applied to the birds are gradually disturbed and die away, and the effect becomes more severe for larger swarm sizes. In addition, the study from Potts et al. also suggests that the manoeuvre wave travels at an increasing speed within the flock (Potts, 1984). This means the memory stored by the birds close to the end of the flock would die away faster, which could also be a factor that limits the memory capacity.

For the ants model, the memory capacity is restricted by the need to adapt to new food sources. The ants might stick around at the path to the previous few food source but would not stick around for too long. This is because the pheromone trail will eventually evaporate and the velocity component for food will gradually drive them to new food sources.

4.4 Variation of Computation Capacity with Swarm Sizes

Figure 4.6 shows the variation of computation capacity with swarm number for both types of reservoirs, before and after applying the PDF observation layer.



(a) (b)

Fig. 4.6 (a) Variation in computation capacity with size of boids reservoir. Covariance of the one with no observation layer applied peaks at 0.11 when swarm number is 22. The performance then drops with swarm number due to permutation symmetry. The application of observation layer solves permutation symmetry, with best covariance at around 0.16. (b) (a) Variation in computation capacity with size of ants reservoir. Covariance of the one with no observation layer applied peaks at 0.11 when swarm number is 24. The performance then drops with swarm number due to permutation symmetry. The application of observation layer solves permutation symmetry, with best covariance at around 0.19.

Looking at the performance before the application of the observation layer, both models are showing similar trends for all the tasks implemented. The performance initially improves with an increase in the number of nodes, reaching an optimal level at a specific threshold. Beyond this point, further additions of nodes result in a decrease in performance. The optimal number for boids and ants are found at 22 and 24, both reaching covariance of around 0.11. This contradicts with ESN, whose performance increases with the number of nodes consistently without dropping.

With the application of the observation layer that solves permutation symmetry, it can be seen that the dropping trend has disappeared for both reservoirs. The covariance now increases with the number and gradually converges to around 0.16 and 0.19 respectively. The results suggest that the main reason for the dropping trend is related to perturbation symmetry. As the swarm number increases, each agent will have more agents to switch their position with. This makes the permutation symmetry issue more severe. The PDF observation layer observes the states of the entire swarm system as a whole, by examining the overall distribution. Therefore, individual swarm agents switching positions during the flight would not affect the output of the observation layer. With the application of this observation layer, we observe that the dropping trend has disappeared.

4.5 Computation Capacity with Different Combinations of Swarm Systems

The heatmap in Figure 4.7 shows the performance of the combined model. The x-axis represents the total number of swarm agents utilised in the reservoir, and the y-axis shows the ratio of combination. The covariance value is represented by the colour of the heatmap, where a brighter colour would indicate a higher covariance value.

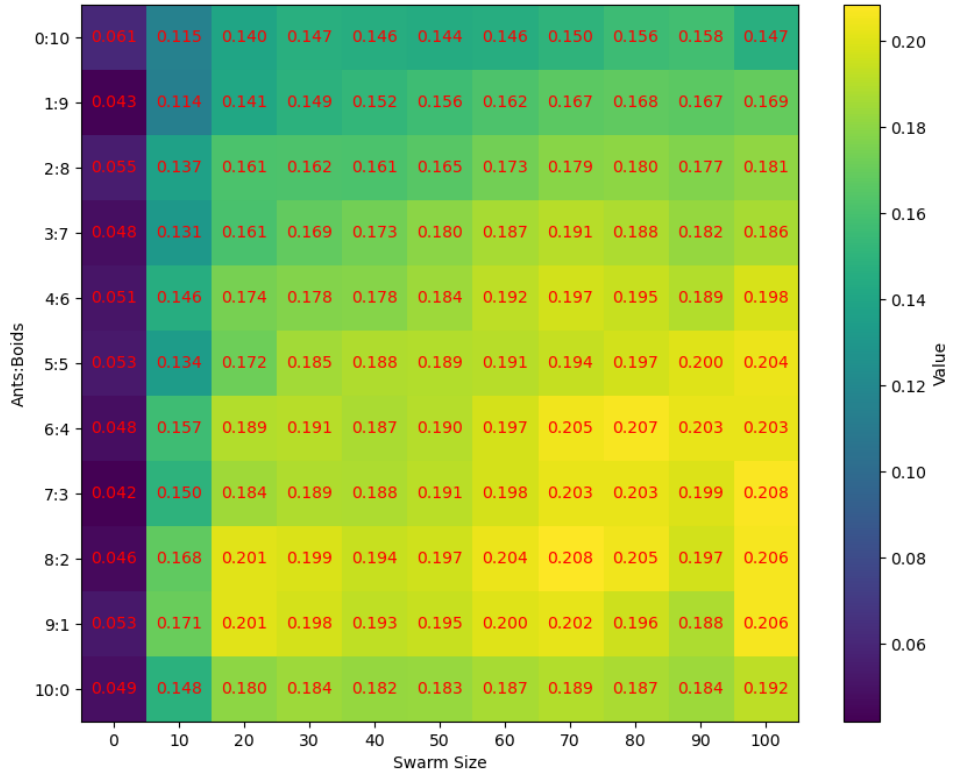


Fig. 4.7 Performance for combined reservoir with different combination ratios. The computation capacity of combined reservoir is generally better than those using a single type of swarm. The combination ratio of ants:boids=8:2 achieves best result among all, with covariance reaching 0.208.

It can be seen that all combinations are showing a similar trend, with the covariance increasing with the total swarm number and gradually converging to a certain level. The result suggests that the performance is at a relatively higher level when the combination ratio

of ants:boids is between 6:4 to 9:1. The ratio of 8:2 would produce a slightly better result among all, with covariance reaching 0.208 when the total swarm number is 70.

Generally speaking, the combined reservoirs show better computation capacities than single swarm reservoirs (when ratio is at 10:0 and 0:10). This improvement comes from the two swarm reservoirs specialising in different tasks. The result verifies our assumption, that combining two different swarm reservoirs would harness the strengths from both of them.

However, we argue that the improvement is still not significant enough compared to using a single type of swarm as reservoir. This is because the two swarm reservoir still exhibits some similarities in computation capacity, such as the weakness in memory capacity suggested in Section 5.3. Potentially, if we can find a swarm reservoir that has a strong memory capacity, combining it with our reservoirs would bring more significant benefits.

4.6 Sensitivity Analysis

A range of sensitivity analyses are conducted according to Table 4.1 and 4.2. Here, the significant ones among them are presented. Other sensitivity analyses of model parameters can be found in Appendix A, and an example of sensitivity analysis on task delays can be found in Appendix B.

4.6.1 Variation of Computation Capacity with Input Ratio (Boids Reservoir)

The input ratio indicates the proportion of birds that directly receive the input. From the plot of input ratio against covariance in Figure 4.8. It can be seen that the covariance improves from 0.04 to 0.15 as the input ratio increases from 0 to 0.5. The performance then stays at a similar level from the input ratio of 0.5 to 0.9, before slightly dropping to 0.14 when the input ratio gets to 1.

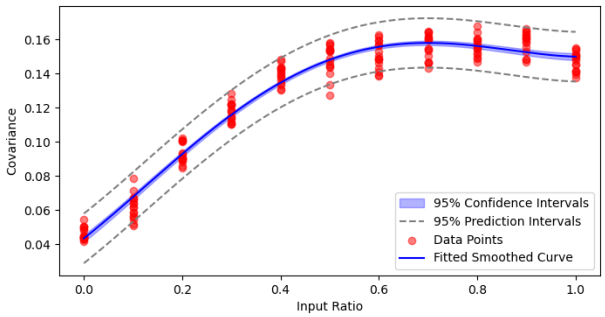


Fig. 4.8 Plot of covariance against input ratio for boids reservoir. Input ratio represents the ratio of birds that directly receive input in the flock.

The result indicates that the performance would degrade when less amount of birds are receiving the input. From Section 2.4.1, the acceleration of each individual bird is affected by its nearby flockmates. If only a small portion of its flockmates are receiving the input, the influence from input might not be strong enough for it to follow, reducing the computation capacity.

On the other hand, when the input ratio is high, a large portion of flockmates are receiving the input. Although the bird can now respond to input correctly, the influence from input might be too strong. This makes each bird follow the input tightly such that the amount of delay among each individual birds is reduced. This reduces the memory capacity of the reservoir.

The input ratio between 0.5 and 0.9 is where these trade-offs are balanced. For the input ratio of 0.5, although the system is having a significant amount of delay, the birds may not follow the input in a precise manner. By comparison, changing the input ratio to 0.9 would reduce the amount of delay but improve the response to input, leading to a similar performance to the case where the ratio is 0.5.

4.6.2 Variation of Computation Capacity with Separation Radius (Boids Reservoir)

Figure 4.9 shows the effect of varying separation radius on computation capacity. It can be seen that the performance is generally well for a separation radius below 40, but drops significantly as the radius gets large.

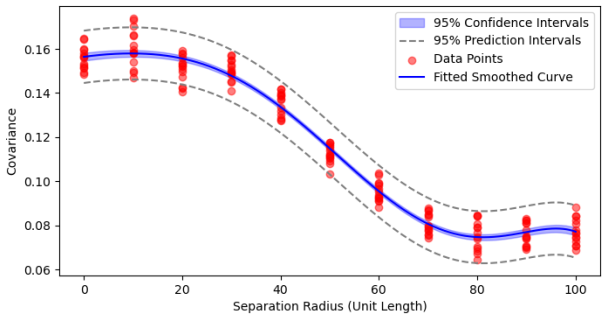


Fig. 4.9 Plot of covariance against separation radius for boids reservoir. The separation radius represents the minimum distance that the birds try to keep from others.

As discussed in Section 3.1.1 , the boids reservoir is a sparsely connected network where all birds only have connections with their nearby flockmates. A larger separation range would reduce the number of birds nearby, making the swarm reservoir network more sparsely connected. This makes the bird less capable of following the input precisely, therefore leading to a reduction in computation capacity. Theoretically, a smaller separation range would make the reservoir network more densely connected, reducing the amount of delay and memory capacity. However, this effect is not significant, as a smaller separation also leads to a better response to input.

4.6.3 Variation of Computation Capacity with Cohesion Radius (Boids Reservoir)

The effect of varying cohesion radius is shown in Figure 4.10. It is interesting to note that when the cohesion range is 0, the boid reservoir actually has a performance that is better than all other values.

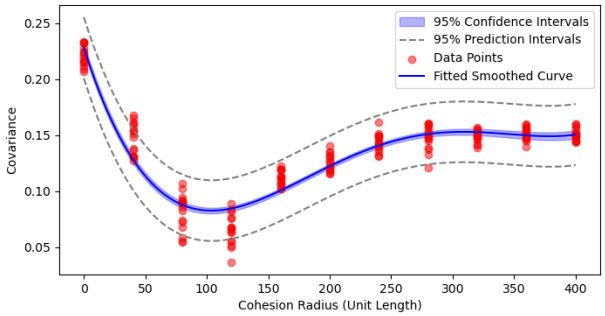


Fig. 4.10 Plot of covariance against cohesion radius for boids reservoir. Birds would try to gather with others within the cohesion radius.

As Section 4.3 mentioned, cohesion force may act as disturbances to the memory of past input states. Therefore, the removal of reduces this kind of disturbance, leading to a better memory capacity. Even without cohesion force, the birds are still flying in one single flock, as the alignment force makes them fly in similar directions. When the cohesion radius gets around 100, we observe the occurrence of local flocking behaviour, where birds are flying in two or more separate flocks. Some of the flocks may not respond well to input, as they do not contain enough birds receiving input. This makes the performance degrade and covariance drop below 0.1. When the cohesion radius gets above 160, we observe that the birds start to fly in one single flock, responding to input. This leads to improvements in computation capacity.

Potentially, we could build a well-performing boids reservoir by ignoring the natural rules and manually setting cohesion force as 0. However, this is out of the scope of our study as our primary goal is to harness swarm intelligence from nature.

4.6.4 Variation of Computation Capacity with Food/Nest Factor Range (Ants Reservoir)

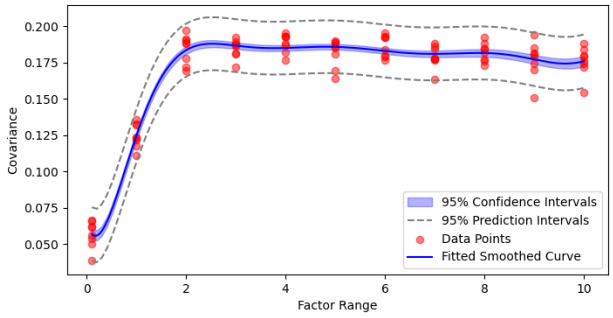


Fig. 4.11 Plot of covariance against food/nest factor range for ants reservoir. This factor influences the impact of food/nest direction to the velocities of ants.

Figure 4.11 shows the computation capacity of ants model at different ranges of food/nest factors. It can be seen that the performance increases as the upper bound of the range varies from 0.1 to 2. This is because a boarder range of food/nest factors increases the diversity in ants' behaviour, giving them a variety of delays that favour reservoir computing. When the upper bound of the range gets above 2, the performance remains at a relatively steady level. This means that the diversity in behaviour also remains at a similar level.

4.7 Chapter Summary

In brief, this chapter illustrates the performance of our RC approach. The effectiveness of our observation layer in solving permutation symmetry is proven. We have also shown that combining different swarm reservoirs would yield better performance than using single swarm reservoirs. The swarm reservoir that yields the best performance is the one that combines ants and boids at the ratio of 8:2. At a swarm size of 20, the computation capacity of our RC framework with this reservoir reaches the level of ESN16. When the swarm size gets above 60, the performance gets to the level of ESN18. This section also shows some sensitivity analysis that helps us gain a deeper understanding of how the system parameters of the swarm systems impact computation capacity.

Chapter 5

Conclusion

In this work, we develop a reservoir computing framework using scalable mobile swarm networks, where swarm models of boids and ants are utilised as our reservoir. The architecture of our swarm network evolves automatically with time. This differs from the traditional reservoirs in reservoir computing which uses a fixed network, but is a more natural way to evaluate how intelligence evolves within the flocks in nature. We found that due to the high permutation symmetry of this mobile network, its performance degrades as network size increases. Therefore, to address the permutation symmetry issue, we develop an observation layer based on Gaussian kernel density estimation, which observes the overall distribution of states and makes the reservoir states permutation invariant.

Our models are tested on four benchmark tasks including short-term memory, parity check, NARMA and Markey-Glass. We illustrate the scalability of our reservoir model, with the performance increases with the size of the reservoir network. In addition, we look into the effect of combining the two types of reservoirs in parallel and proved that harnessing dynamics from two different swarm reservoirs can bring improvements to computation capacity. Our best model is the one that combines ants and boids reservoir at a ratio of 8:2. For a swarm size of 20, the performance reaches a covariance around 0.2 which is equivalent to ESN16. Then when the swarm number gets above 60, a covariance value of 0.208 is reached, which is equivalent to ESN18. This means that our swarm reservoir is able to perform simple computation tasks. Sensitivity analyses are also conducted to investigate the effect of varying parameters in the swarm reservoirs. Interestingly, we discover that setting the cohesion force of the boids reservoir to zero yields a better result, as it reduces the disturbance of memories of past inputs.

5.1 Future Works

The performance of our model still has a large gap with ESNs of larger sizes, which remains to be improved for future work. This is because the dynamics of the Boids and Ants model we use only allow memory for the past few states, which limits the computation capacity of the reservoir. Therefore, a more suitable swarm reservoir that exhibits a stronger memory capacity is needed to be found. This new reservoir could potentially be established based on our model, with some modifications to certain dynamics and parameters, which remains an interesting aspect for investigation.

Another potential future work for this project is to apply our reservoir computing framework to swarm systems in real life. There might be moral concerns about applying it to living swarm systems in nature, but it can still be applied to artificial swarm systems such as drones. Our framework allows the collective intelligence within these mobile swarm systems to be transformed into significant computational resources, offering considerable computational power that benefits society.

References

Baldini, P. (2022). Reservoir computing in robotics: a review. *arXiv preprint arXiv:2206.11222*.

Beckers, R., Deneubourg, J.-L., Goss, S., and Pasteels, J. M. (1990). Collective decision making through food recruitment.

Chevallier, S., Paugam-moisy, H., Sebag, M., et al. (2010). Spikeants, a spiking neuron network modelling the emergence of organization in a complex system. *Advances in Neural Information Processing Systems*, 23.

Deneubourg, J.-L. and Goss, S. (1989). Collective patterns and decision-making. *Ethology Ecology & Evolution*, 1(4):295–311.

Dorigo, M., Birattari, M., and Stutzle, T. (2006). Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39.

Dussutour, A., Nicolis, S. C., Shephard, G., Beekman, M., and Sumpter, D. J. (2009). The role of multiple pheromones in food recruitment by ants. *Journal of Experimental Biology*, 212(15):2337–2348.

Emlen, J. T. (1952). Flocking behavior in birds. *The Auk*, 69(2):160–170.

Grüter, C., Czaczkes, T. J., and Ratnieks, F. L. (2011). Decision making in ant foragers (lasius niger) facing conflicting private and social information. *Behavioral Ecology and Sociobiology*, 65:141–148.

Holzmann, G. and Hauser, H. (2010). Echo state networks with filter neurons and a delay&sum readout. *Neural Networks*, 23(2):244–256.

Isaeva, V. (2012). Self-organization in biological systems. *Biology Bulletin*, 39:110–118.

Jaeger, H. (2001). The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13.

Jaeger, H. and Haas, H. (2004). Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. ieee.

Kvam, O. V. and Kleiven, O. T. (1995). Diel horizontal migration and swarm formation in daphnia in response to chaoborus. In *Cladocera as Model Organisms in Biology: Proceedings of the Third International Symposium on Cladocera, held in Bergen, Norway, 9–16 August 1993*, pages 177–184. Springer.

Lissaman, P. B. and Shollenberger, C. A. (1970). Formation flight of birds. *Science*, 168(3934):1003–1005.

Lowe, D. and Broomhead, D. (1988). Multivariable functional interpolation and adaptive networks. *Complex systems*, 2(3):321–355.

Lukoševičius, M. (2012). A practical guide to applying echo state networks. In *Neural Networks: Tricks of the Trade: Second Edition*, pages 659–686. Springer.

Lymburn, T., Algar, S. D., Small, M., and Jüngling, T. (2021). Reservoir computing with swarms. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 31(3).

Mataric, M. J. (1992). Designing emergent behaviors: From local interactions to collective intelligence. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 432–441.

Nakajima, K. (2020). Physical reservoir computing—an introductory perspective. *Japanese Journal of Applied Physics*, 59(6):060501.

Nakajima, K., Hauser, H., Kang, R., Guglielmino, E., Caldwell, D. G., and Pfeifer, R. (2013). A soft body as a reservoir: case studies in a dynamic model of octopus-inspired soft robotic arm. *Frontiers in computational neuroscience*, 7:91.

Nakajima, K., Hauser, H., Li, T., and Pfeifer, R. (2015). Information processing via physical soft body. *Scientific reports*, 5(1):10487.

Nakajima, K., Li, T., Hauser, H., and Pfeifer, R. (2014). Exploiting short-term memory in soft body dynamics as a computational resource. *Journal of The Royal Society Interface*, 11(100):20140437.

Potts, W. K. (1984). The chorus-line hypothesis of manoeuvre coordination in avian flocks. *Nature*, 309(5966):344–345.

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34.

Silverman, B. W. (2018). *Density estimation for statistics and data analysis*. Routledge.

Stanley, K. O., Clune, J., Lehman, J., and Miikkulainen, R. (2019). Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.

Sumpter, D. J. (2010). *Collective animal behavior*. Princeton University Press.

TAKAGAKI, S., TATEISHI, K., and ANDO, H. (2024). " reservoir computing"–time-series prediction method for fast and accurate soft sensor modeling. *Mitsubishi Heavy Industries Technical Review*, 61(1):1.

Tanaka, G., Yamane, T., Héroux, J. B., Nakane, R., Kanazawa, N., Takeda, S., Numata, H., Nakano, D., and Hirose, A. (2019). Recent advances in physical reservoir computing: A review. *Neural Networks*, 115:100–123.

Wang, X. and Cichos, F. (2024). Harnessing synthetic active particles for physical reservoir computing. *Nature Communications*, 15(1):774.

Wierstra, D., Schaul, T., Glasmachers, T., Sun, Y., Peters, J., and Schmidhuber, J. (2014). Natural evolution strategies. *The Journal of Machine Learning Research*, 15(1):949–980.

Zhao, Q., Nakajima, K., Sumioka, H., Hauser, H., and Pfeifer, R. (2013). Spine dynamics as a computational resource in spine-driven quadruped locomotion. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1445–1451. IEEE.

Appendix A

Other Sensitivity Analysis Performed

A.1 Variation of Computation Capacity with Alignment Radius (Boids Reservoir)

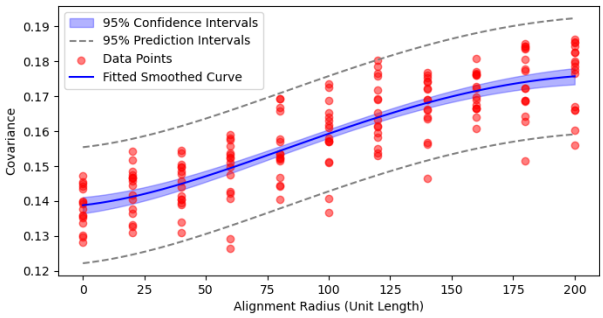


Fig. A.1 Plot of covariance against alignment radius for boids reservoir. Alignment radius indicates the range of neighbours that the birds match their velocities with.

The relationship between alignment radius and performance is also evaluated and shown in Figure A.1, where the computation capacity increases with alignment radius. This behaviour is related to the trade-off that Section 5.6.1 mentions. When the alignment radius is small, each individual birds only aligns with its neighbours. This makes the system have significant amounts of delay, but also causes some of the birds not to follow the input tightly. When the alignment radius is large, each individual bird would align with more flockmates, including those directly receiving inputs. As a result, the system would become more stable as birds follow input more tightly, despite the reduction in the amount of delay. The result shows the effect of following input more precisely outweighs the reduction in delay, as the computation capacity increases with alignment radius.

However, this does not suggest that having a larger alignment range would produce a better boids reservoir. As discussed in Section 5.3, the past input states of the boids reservoir are significantly disturbed and die away very fast, due to the complex dynamics and interactions among the swarm system. For a small alignment radius, although the system exhibits a significant amount of delay, retaining memory of many past input states, these memories are significantly disturbed. For a large alignment radius, although the system only has memories of the past few input states, the disturbance to the states is smaller as the forces of alignment become stronger. Potentially, if we can find a way to reduce the disturbance to states, a smaller alignment radius could yield a better result.

A.2 Variation of Computation Capacity with Alignment Factor (Boids Reservoir)

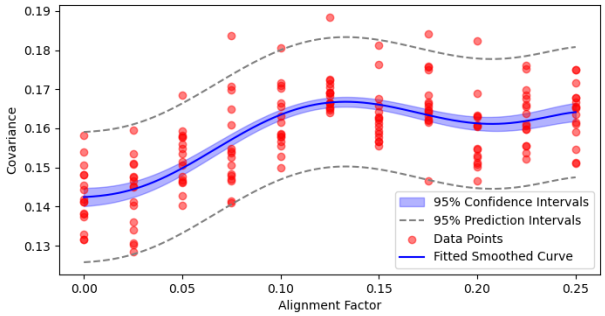


Fig. A.2 Plot of covariance against alignment factor for boids reservoir. Alignment factor indicates the strength of influence of alignment forces on birds' velocities

Figure A.2 shows the variation of computation capacity for different values of alignment factor. It can be seen that the covariance is at a lower level when the factor is below 0.05. This is because the alignment force is not strong enough for the birds to follow the input angle precisely. When the factor is above 0.05, the performance seems to be fluctuating around a comparable level. The alignment factor is directly related to the amount of alignment force on the birds. A larger alignment factor would result in a better response to input with a reduction in the amount of delay, while a smaller alignment factor would have the opposite effect. In this scenario, it seems that the two effects have similar influences on computation capacities, as the covariance is at a comparable level for all alignment factors above 0.05.

A.3 Variation of Computation Capacity with Separation Factor (Boids Reservoir)

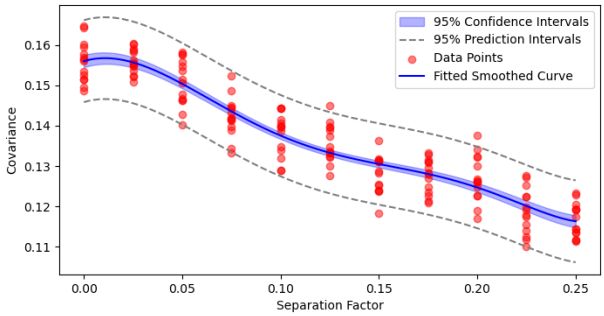


Fig. A.3 Plot of covariance against separation factor for boids reservoir. Separation factor indicates the strength of influence of Separation forces on birds' velocities

Figure A.3 shows the effect of varying separation factors to computation capacity. It can be seen that the performance is generally well for a separation factor below 0.05, but drops significantly as the value gets large. The separation factor is directly related to the amount of separation force on the birds. When the separation force is large, the birds would immediately make a significant change in the direction of flight when any other bird enters their separation radius. This makes the state of the bird very unstable and fails to follow input in a precise manner. Theoretically, a smaller separation force would make the birds get closer to each other, reducing the amount of delay and memory capacity. However, this effect is not significant, as a smaller separation also ensures a better response to input.

A.4 Variation of Computation Capacity with Cohesion Factor (Boids Reservoir)

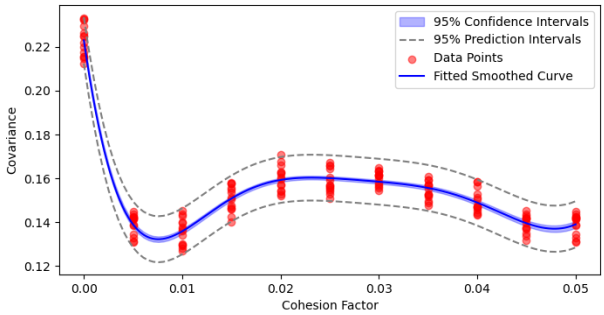


Fig. A.4 Plot of covariance against cohesion factor for boids reservoir. Cohesion factor indicates the strength of influence of cohesion forces to birds' velocities

The effect of varying cohesion factors is shown in Figure A.4. When the cohesion factor is 0, the cohesion force on birds also becomes 0. As previously discussed in the analysis of cohesion radius in Section 5.6.3, the absence of cohesion force reduces the disturbance on the memory of past input states within the flock, leading to a better memory capacity. The performance significantly degrades as the cohesion factor becomes non-zero. The performances for the cohesion factor within the range of 0.015 to 0.04 are slightly better than others. This indicates that within this range, the birds would fly in one single group following the input properly, with suitable amount of delays among them.

Appendix B

Sensitivity Analysis in Delays

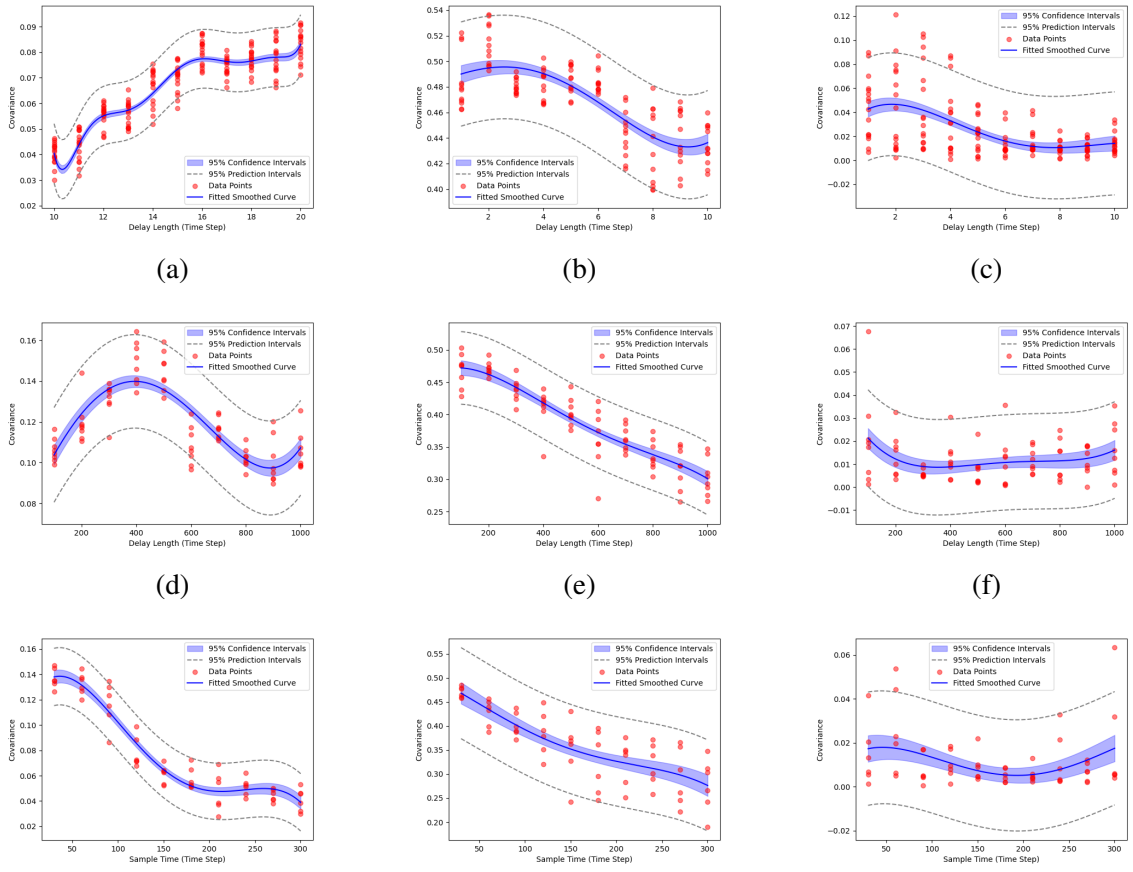


Fig. B.1 Plots of delay analysis for swarm reservoirs of size 40. (a) Plot of covariance against delay length for boids reservoir in STM and PC tasks. (b) Plot of covariance against delay length for boids reservoir in NARMA tasks. (c) Plot of covariance against delay length for boids reservoir in MG tasks. (d) Plot of covariance against delay length for ants reservoir in STM and PC tasks. (e) Plot of covariance against delay length for ants reservoir in NARMA tasks. (f) Plot of covariance against delay length for boids reservoir in MG tasks. (g) Plot of covariance against sample time for ants reservoir in STM and PC tasks. (h) Plot of covariance against sample time for ants reservoir in NARMA tasks. (i) Plot of covariance against sample time for ants reservoir in MG tasks.

Figure B.1 shows an example of how performance in different amounts of delays are evaluated and selected. We conduct this kind of delay analysis for several sizes of swarms, and select the set of delay that is most suitable for all sizes of swarms.