

# Graph Neural Network-based Approximate Bayesian Computation for Agent-based Model Calibration of Bacterial Population Growth

by

Xianglong Bai

A thesis  
presented to the University of Waterloo  
in fulfillment of the  
thesis requirement for the degree of  
Master of Mathematics  
in  
Applied Mathematics

Waterloo, Ontario, Canada, 2026

© Xianglong Bai 2026

### **Author's Declaration**

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

## Abstract

Approximate Bayesian Computation (ABC) has emerged as a powerful likelihood-free inference framework for model selection and parameter inference in complex biological systems where explicit likelihood functions are intractable or computationally prohibitive.

However, the effectiveness of ABC strongly depends on the choice of summary statistics and distance metrics used to compare simulated and observed data. When analyzing time-lapse observations of growing cell populations, the selection of suitable summary statistics often relies on manually designed features informed by domain expertise. Designing such statistics is challenging as they must capture complex spatial, structural, and temporal characteristics of the biological system. Consequently, handcrafted summary statistics may omit relevant information or fail to generalize across datasets. As a result, important information contained in the data may be lost, potentially leading to inefficient inference or biased posterior estimates. This motivates the use of deep learning approaches, such as Graph Neural Networks (GNNs), which can automatically learn informative representations directly from graph-structured data.

To address these limitations, this thesis proposes and systematically investigates four novel strategies for integrating deep learning approaches into the Sequential Monte Carlo ABC (ABC-SMC) framework, with a focus on GNNs and Long Short-Term Memory (LSTM) models. These architectures are specifically designed to capture the relational structure of cell populations and the temporal dynamics inherent in time-lapse data.

Using GNNs, we encoded spatial interactions among cells through contact edges in graph representations of the biological system. The temporal dynamics of the evolving cell population are captured in two ways. In one approach, LSTM layers are incorporated to model dependencies across successive graph observations in time-lapse sequences. In the alternative approach, we represent temporal relationships directly within the graph structure through lineage edges in a knowledge graph, which explicitly encode parent-daughter relationships between cells over time.

We consider two learning paradigms for extracting informative representations from these graphs. In the first approach, graph regression models are trained using mean squared error (MSE) to directly predict model parameters from simulated data. In the second approach, graph embedding models are trained with a triplet loss to learn low-dimensional representations that preserve the similarity relationships among simulations generated from similar parameter configurations.

The resulting representations serve as GNN-based summary statistics, replacing conventional handcrafted statistics within the ABC-SMC inference pipeline. Such deep learning

approaches belong to the broader class of GNN-based methods for likelihood-free inference, which aim to automatically extract informative features from complex simulation outputs.

We evaluate the proposed strategies against a baseline approach relying on classical summary statistics. Inference performance is assessed using two complementary metrics. One is the Kullback–Leibler (KL) divergence between the inferred posterior distributions and the ground-truth parameters. The other is the mean squared distance (MSD) between the inferred and true parameter values. Across all evaluated strategies, the GNN-based summary statistics consistently outperform conventional handcrafted summary statistics for simulation studies. They yield more accurate posterior approximations, as reflected by reduced KL divergence, and more precise parameter estimates, as reflected by lower MSD values. However, the results are less convincing on real data, likely due to model mismatch.

Overall, this work demonstrates that replacing handcrafted summary statistics with GNN-based ones can substantially improve likelihood-free inference in complex biological systems, assuming that there is no model mismatch and no unknown noise in observations from real experiments. By integrating GNNs with the ABC–SMC framework, the proposed approach enables the automatic extraction of informative representations from graph-structured, time-evolving population data. The resulting methodology provides a principled strategy for parameter inference, bridging computational simulations and experimental observations through simulation-based model calibration. Although the biological model considered in this study serves primarily as a simple test to develop the inference pipeline, the proposed framework is designed to be readily extended to more complex cases commonly encountered in systems biology.

## Acknowledgements

I first express my sincere gratitude to my supervisor, Professor Brian Ingalls, for his patience, guidance, and continuous support throughout my research. Despite my limited background in the biological aspects of this project, he consistently took the time to explain concepts clearly and encouraged me to develop my own research approach. He was always understanding of my mistakes and maintained a positive and open attitude, welcoming discussions of all questions, no matter how trivial they might have seemed. Our regular meetings (often at least once, and sometimes multiple times each week) provided invaluable guidance and ensured that I stayed on the right track.

I would like to thank Atiyeh Ahmadi for her kind support and for providing valuable resources that greatly contributed to the progress of my research. Her encouragement was especially meaningful during times when I faced challenges in both my studies and my personal life. I sincerely believe that she will make an excellent professor in the future, as her kindness and support would undoubtedly benefit her students.

I would also like to thank Aaron Yip for his assistance with the simulation aspects of this project and for identifying and resolving issues in my code. His expertise and willingness to help were greatly appreciated. I hope that one day I can approach biological modeling with the same level of skill and insight as he demonstrates.

Lastly, I am deeply grateful to Kunal Ghosh, Gaia Noseworthy, Maryam Yalsavar, Alaa Elsayed, Peiyi Zheng, Pengfei Yue, Yuheng Wen, Nan Liu and Venkatesh Sangiah Karuppiah for being by my side during the preparation of this thesis and for their constant encouragement and emotional support.

## **Dedication**

I would like to thank Tao, Lin, and Yanqiu for their unconditional love and support throughout my studies and in my life.

Last but not least, I would like to thank Professor Brian Ingalls once again for the opportunity to be here, which has allowed me to formally begin my journey in Applied Mathematics and Applied Statistics research, as I had hoped.

# Table of Contents

|                                                                     |              |
|---------------------------------------------------------------------|--------------|
| <b>Author's Declaration</b>                                         | <b>ii</b>    |
| <b>Abstract</b>                                                     | <b>iii</b>   |
| <b>Acknowledgements</b>                                             | <b>v</b>     |
| <b>Dedication</b>                                                   | <b>vi</b>    |
| <b>List of Figures</b>                                              | <b>xi</b>    |
| <b>List of Tables</b>                                               | <b>xviii</b> |
| <b>1 Introduction</b>                                               | <b>1</b>     |
| 1.1 Context Background . . . . .                                    | 1            |
| 1.2 Agent-based Model . . . . .                                     | 2            |
| 1.2.1 Definition . . . . .                                          | 2            |
| 1.2.2 Advantages of Simulation . . . . .                            | 3            |
| 1.2.3 Difficulties of Calibration . . . . .                         | 5            |
| 1.3 Deep Learning Preliminaries . . . . .                           | 8            |
| 1.3.1 Neural Network and Basics . . . . .                           | 8            |
| 1.3.2 Recurrent Neural Network and Long Short-term Memory . . . . . | 13           |
| 1.3.3 Graph Neural Network and Multi-relational Graph . . . . .     | 22           |

|          |                                                                                           |           |
|----------|-------------------------------------------------------------------------------------------|-----------|
| 1.3.4    | Activation Function in Graph Neural Network . . . . .                                     | 27        |
| 1.3.5    | Graph Convolutional Network and Graph Attention Network . . . . .                         | 29        |
| 1.3.6    | Hyperparameter Tuning of Graph Attention Network . . . . .                                | 31        |
| 1.3.7    | Over-smoothing Issues and Dropout Approaches in Graph Neural<br>Network . . . . .         | 34        |
| 1.3.8    | Batch Size . . . . .                                                                      | 37        |
| 1.4      | Experimental Setup . . . . .                                                              | 38        |
| 1.5      | Simulation Module Setup . . . . .                                                         | 38        |
| 1.6      | Conventional Approximate Bayesian Computation . . . . .                                   | 41        |
| 1.6.1    | Bayesian Inference . . . . .                                                              | 41        |
| 1.6.2    | Approximate Bayesian Computation Sequential Monte Carlo . . . . .                         | 42        |
| 1.6.3    | Importance Weights . . . . .                                                              | 45        |
| 1.6.4    | Perturbation Kernel and Effective Sample Size . . . . .                                   | 47        |
| 1.6.5    | Distribution-based Distance with Kullback-Leibler Divergence . . . . .                    | 48        |
| 1.6.6    | Previous Work . . . . .                                                                   | 51        |
| 1.7      | Semi-automatic and Machine Learning-based Approximate Bayesian Com-<br>putation . . . . . | 52        |
| 1.8      | Informativeness of Summary Statistics . . . . .                                           | 54        |
| 1.9      | Dimensionality and Interpretability of Summary Statistics . . . . .                       | 56        |
| 1.9.1    | Dimensionality of Summary Statistics . . . . .                                            | 56        |
| 1.9.2    | Interpretability of Summary Statistics . . . . .                                          | 57        |
| 1.9.3    | Dimensionality of Machine Learning-based Summary Statistics . . . . .                     | 58        |
| 1.9.4    | Interpretability of Machine Learning-based Summary Statistics . . . . .                   | 60        |
| 1.10     | Problems and Objectives . . . . .                                                         | 60        |
| <b>2</b> | <b>Methods</b>                                                                            | <b>62</b> |
| 2.1      | Datasets and Features . . . . .                                                           | 62        |
| 2.1.1    | Data Formats . . . . .                                                                    | 62        |

|       |                                                                                                                                         |     |
|-------|-----------------------------------------------------------------------------------------------------------------------------------------|-----|
| 2.1.2 | Parameter Value Selection for Training Data . . . . .                                                                                   | 63  |
| 2.1.3 | Data Cleaning and Preprocessing . . . . .                                                                                               | 66  |
| 2.1.4 | Feature Importance Ranking . . . . .                                                                                                    | 66  |
| 2.2   | Simulated-experimental Data . . . . .                                                                                                   | 68  |
| 2.3   | Pipeline Overview . . . . .                                                                                                             | 68  |
| 2.4   | Proposed Graph Neural Network-based Architectures . . . . .                                                                             | 73  |
| 2.4.1 | Training Setup . . . . .                                                                                                                | 73  |
| 2.4.2 | First Strategy: Graph Neural Network Regression in Approximate Bayesian Computation . . . . .                                           | 74  |
| 2.4.3 | Second Strategy: Graph Neural Network Regression in Approximate Bayesian Computation with Long Short-term Memory . . . . .              | 79  |
| 2.4.4 | Third Strategy: Graph Neural Network Representation Learning in Approximate Bayesian Computation . . . . .                              | 83  |
| 2.4.5 | Fourth Strategy: Graph Neural Network Representation Learning in Approximate Bayesian Computation with Long Short-term Memory . . . . . | 89  |
| 2.4.6 | Fifth Strategy: Conventional Approximate Bayesian Computation Method . . . . .                                                          | 93  |
| 2.4.7 | Summary . . . . .                                                                                                                       | 94  |
| 2.5   | Posterior Comparisons over Strategies . . . . .                                                                                         | 95  |
| 2.5.1 | Delta Function and Kernel Density Estimation . . . . .                                                                                  | 95  |
| 2.5.2 | Gaussian Approximation of a Dirac Delta . . . . .                                                                                       | 96  |
| 2.5.3 | Simulation-based Validation in Approximate Bayesian Computation . . . . .                                                               | 99  |
| 2.5.4 | Effect of Unknown Noise and Model Mismatch . . . . .                                                                                    | 100 |
| 2.5.5 | Informativeness Measure by Simulation-based Validation Results . . . . .                                                                | 102 |
| 2.5.6 | Mean Squared Error and Squared Error . . . . .                                                                                          | 106 |
| 2.5.7 | Replacement of Simulated-experimental Data with Real-experimental Data . . . . .                                                        | 107 |

|          |                                                                                             |            |
|----------|---------------------------------------------------------------------------------------------|------------|
| <b>3</b> | <b>Results</b>                                                                              | <b>109</b> |
| 3.1      | Feature Importance Ranking . . . . .                                                        | 109        |
| 3.2      | Feature Selection . . . . .                                                                 | 112        |
| 3.3      | Parameter Predictions . . . . .                                                             | 115        |
| 3.4      | Data Subset Ratio . . . . .                                                                 | 117        |
| 3.5      | Result Comparison between Strategies . . . . .                                              | 118        |
| 3.5.1    | Training Results . . . . .                                                                  | 118        |
| 3.5.2    | Posterior Inference Result Comparisons between Strategies . . . . .                         | 120        |
| 3.5.3    | Posterior Inference Results for Strategies . . . . .                                        | 121        |
| 3.6      | Conventional Inference Result with Observations of Real Experiments . . . . .               | 125        |
| 3.7      | Graph Neural Network-based Inference Result with Observations of Real Experiments . . . . . | 127        |
| 3.8      | Issue with Node Connectivity in Graphs and Model Discrepancy . . . . .                      | 131        |
| 3.9      | Future Work . . . . .                                                                       | 133        |
| 3.10     | Conclusion . . . . .                                                                        | 137        |
|          | <b>References</b>                                                                           | <b>139</b> |
|          | <b>APPENDICES</b>                                                                           | <b>145</b> |
| <b>A</b> | <b>More Details about Data and Features</b>                                                 | <b>146</b> |
| <b>B</b> | <b>Results of Previous Case Study</b>                                                       | <b>153</b> |

# List of Figures

|      |                                                                                                                                                                                                                                                                                                                                                                                                     |    |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 1.1  | Neural Network Architecture. This figure is obtained from <a href="https://www.geeksforgeeks.org/deep-learning/artificial-neural-networks-and-its-applications/">https://www.geeksforgeeks.org/deep-learning/artificial-neural-networks-and-its-applications/</a> .                                                                                                                                 | 13 |
| 1.2  | LSTM cell architecture <a href="https://www.geeksforgeeks.org/deep-learning/deep-learning-introduction-to-long-short-term-memory/">https://www.geeksforgeeks.org/deep-learning/deep-learning-introduction-to-long-short-term-memory/</a> .                                                                                                                                                          | 20 |
| 1.3  | Comparing different Sequence models: RNN, LSTM. This figure is obtained from <a href="https://colah.github.io/posts/2015-08-Understanding-LSTMs/">https://colah.github.io/posts/2015-08-Understanding-LSTMs/</a> .                                                                                                                                                                                  | 22 |
| 1.4  | A Multiplex Graph with 2 edge types [25].                                                                                                                                                                                                                                                                                                                                                           | 24 |
| 1.5  | Information in the form of scalars or embeddings can be stored at each graph node (left) or edge (right) [50].                                                                                                                                                                                                                                                                                      | 26 |
| 1.6  | Comparison of ReLU, Leaky ReLU, and ELU activation functions. ReLU maps all negative inputs to zero, Leaky ReLU preserves a small linear slope in the negative region, and ELU produces smooth negative outputs. Both Leaky ReLU and ELU help mitigate the vanishing-gradient behavior associated with inactive ReLU units.                                                                         | 27 |
| 1.7  | Attention on a graph node [30].                                                                                                                                                                                                                                                                                                                                                                     | 31 |
| 1.8  | Illustration of a DropEdge for network learning [27].                                                                                                                                                                                                                                                                                                                                               | 36 |
| 1.9  | Experimental samples of Time-lapse images from the first frame to the last frame [13]. This figure is from Sushma Dhamodharan, one of the previous master's students in our research group. The experimental images were taken by my senior, Atiyeh Ahmadi [2].                                                                                                                                     | 38 |
| 1.10 | Simulated sample images from timestep $t = t_1$ to timestep $t = t_N$ . This visualization mimics time-lapse images from real experiments. The timestep $t_1 < t_2 < t_3 < t_N$ , but it's not assumed that $t_{(i-1)} + 1 = t_i$ for all the $i$ from 1 to $N$ . This means that these four images are arbitrarily selected from the simulation videos and are not necessarily sequential in time. | 39 |

|      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |    |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 1.11 | ABM parameters $\alpha$ and $\gamma$ , the left side indicates when $\alpha$ and $\gamma$ are low, the right side shows how the displacement and growth affect the cells and their neighbors as the value increases [13]. This figure is from Sushma Dhamodharan, one of the previous master's students in our research group.                                                                                                                                                                                                                                                          | 40 |
| 1.12 | ABM parameters $\alpha$ (left) and $\gamma$ (right). This figure was done by my seniors.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 40 |
| 1.13 | Example of Bayesian inference with a prior distribution, a posterior distribution, and a likelihood function. The prediction error is the difference between the prior expectation and the peak of the likelihood function (i.e., reality). Uncertainty is the variance of the prior. Noise is the variance of the likelihood function. [61]. . . . .                                                                                                                                                                                                                                   | 42 |
| 2.1  | Different data formats in different stages of the project. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | 63 |
| 2.2  | Proposed distributions for parameter selection with different numbers of samples. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | 65 |
| 2.3  | A pipeline overview flowchart. The arrows highlighted in red denote a simulator circulation or a circulation of a simulator set with a fixed pair of parameters (i.e., gamma and alpha). From left to right, the first decision gate compares simulated summary statistics with experimental summary statistics; the second decision gate determines whether the population size is enough to finish the current generation; the third decision gate determines whether the current generation adapts to the next one with a lower acceptance threshold in the distance metric. . . . . | 69 |
| 2.4  | Cell Lineage and Neighbor [13]. The left side shows how the cell divides; notice that not every cell divides at each time step. At each time step, each node will have a different ID. The right panel shows the layout of the cell at that time step, along with the contact information for the cells. This figure is from Sushma Dhamodharan, one of the previous master's students in our research group. . . . .                                                                                                                                                                   | 72 |
| 2.5  | Graph representation of colony for various intervals [13]. Red lines are contact edges, and blue arrows are directed lineage edges. This figure is from Sushma Dhamodharan, one of the previous master's students in our research group. . . . .                                                                                                                                                                                                                                                                                                                                        | 72 |

|      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |     |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 2.6  | The architecture of the first strategy: GNN regressor in the ABC. This architecture explains what's inside the trained GNN model, as shown in Figure 2.3, which is the pipeline overview flowchart in section 2.3. The output summary statistics are the parameter gamma and alpha predicted by the GNN regressor. The labeled targets are the parameters gamma and alpha as well. . . . .                                                                                                               | 77  |
| 2.7  | The architecture of the second strategy: GNN-LSTM regressor in the ABC. This architecture explains what's inside the trained GNN model Figure 2.3, which is the pipeline overview flowchart in section 2.3. The output summary statistics are the parameter gamma and alpha predicted by the GNN regressor. The labeled targets are the parameters gamma and alpha as well. The GNN regressor architecture is fixed over sequential graphs. . . . .                                                      | 82  |
| 2.8  | Visualization of triplet loss [36]. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                              | 86  |
| 2.9  | The architecture of the third strategy: GNN embedding learner in the ABC. This architecture explains what's inside the trained GNN model in Figure 2.3, which is the pipeline overview flowchart in section 2.3. The output summary statistics are graph-level representations generated by the GNN embedding learner. The triplets are composed of anchor graphs, positive graphs and negative graphs. . . . .                                                                                          | 88  |
| 2.10 | The architecture of the fourth strategy: GNN-LSTM embedding learner in the ABC. This architecture explains what's inside the trained GNN model, as shown in Figure 2.3, which is the pipeline overview flowchart in section 2.3. The output summary statistics are graph-level representations generated by the GNN embedding learner. The triplets are composed of anchor graphs, positive graphs, and negative graphs. The GNN embedding learner architecture is fixed over sequential graphs. . . . . | 93  |
| 2.11 | Gaussian approximation of Dirac delta functions. The orders are $\varepsilon = 0.02$ (top-left), $\varepsilon = 0.05$ (top-right), $\varepsilon = 0.1$ (bottom-left), and $\varepsilon = 0.1$ (bottom-right). . . . .                                                                                                                                                                                                                                                                                    | 98  |
| 2.12 | Sampling validation process flowchart for the first and second strategies. The summary statistics are the parameters themselves in the first and second strategies, so redundant and duplicate steps are removed to improve efficiency. . . . .                                                                                                                                                                                                                                                          | 108 |
| 2.13 | Sampling validation process flowchart for the third and fourth strategies. The summary statistics are the graph embeddings in the third and fourth strategies, so the trained GNNs are applied. . . . .                                                                                                                                                                                                                                                                                                  | 108 |

|      |                                                                                                                                                                                                                                                                                          |     |
|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 2.14 | Sampling validation process flowchart for the fifth strategy. The summary statistics are the traditional ones in this strategy, so the trained GNNs are removed. . . . .                                                                                                                 | 108 |
| 3.1  | GAT-based attention feature importance rank in terms of both contact and lineage edge types. . . . .                                                                                                                                                                                     | 110 |
| 3.2  | GAT-based attention feature importance rank in terms of the contact edge type only. . . . .                                                                                                                                                                                              | 111 |
| 3.3  | GAT-based attention feature importance rank in terms of the lineage edge type only. . . . .                                                                                                                                                                                              | 112 |
| 3.4  | Training results of 6 cases. The orders are 57 features (top-left), 47 features (top-middle), 37 features (top-right), 27 features (bottom-left), 17 features (bottom-middle), and 7 features (bottom-right). . . . .                                                                    | 113 |
| 3.5  | Predictions vs targets of 6 cases. The orders are 57 features (top-left), 47 features (top-right), 37 features (middle-left), 27 features (middle-right), 17 features (bottom-left), and 7 features (bottom-right). . . . .                                                              | 114 |
| 3.6  | Comparison between GNN predictions and targets when the Z-score normalization is applied to both features and targets during the training. Note that targets are gamma and alpha parameters here. . . . .                                                                                | 116 |
| 3.7  | Comparison between GNN predictions and targets when the Z-score normalization is applied to both features and the logarithm base 10 of targets during the training. Note that targets are gamma and alpha parameters here. . . . .                                                       | 116 |
| 3.8  | Comparison between GNN predictions and targets when the Z-score normalization is applied to both features and the logarithm base 10 of targets during the training, but the targets are de-normalized after the training. Note that targets are gamma and alpha parameters here. . . . . | 117 |
| 3.9  | Plots to verify that the target distribution in training/validating/testing sets is close. The targets are in the log space, base e, and are normalized using Z-score normalization. . . . .                                                                                             | 118 |
| 3.10 | Training result comparison between GNN regressor (as described in subsection 2.4.2) and GNN-LSTM regressor (as described in subsection 2.4.3) in the ABC. . . . .                                                                                                                        | 119 |
| 3.11 | Training result comparison between GNN embedding learner (as described in subsection 2.4.4) and GNN-LSTM embedding learner (as described in subsection 2.4.5) in the ABC. . . . .                                                                                                        | 119 |

|      |                                                                                                                                                                                                                                                                          |     |
|------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 3.12 | ABC-SMC posterior results over generations for GNN regressor (as described in subsection 2.4.2).                                                                                                                                                                         | 122 |
| 3.13 | ABC-SMC posterior results over generations for GNN-LSTM regressor (as described in subsection 2.4.3).                                                                                                                                                                    | 122 |
| 3.14 | ABC-SMC posterior results over generations for GNN embedding learner (as described in subsection 2.4.4).                                                                                                                                                                 | 123 |
| 3.15 | ABC-SMC posterior results over generations for GNN-LSTM embedding learner (as described in subsection 2.4.5).                                                                                                                                                            | 123 |
| 3.16 | ABC-SMC posterior results over generations for the conventional method (as described in subsection 2.4.6).                                                                                                                                                               | 124 |
| 3.17 | More helpful ABC-SMC information.                                                                                                                                                                                                                                        | 124 |
| 3.18 | ABC-SMC posterior results over generations, with logarithm base 10 and relative distance.                                                                                                                                                                                | 125 |
| 3.19 | ABC-SMC posterior results over generations, with logarithm base $e$ and relative distance.                                                                                                                                                                               | 126 |
| 3.20 | ABC-SMC posterior results over generations, with logarithm base $e$ and KL divergence.                                                                                                                                                                                   | 126 |
| 3.21 | ABC-SMC posterior sampling validation, with logarithm base $e$ . The left result is based on the relative distance. The right result is based on the KL divergence.                                                                                                      | 127 |
| 3.22 | GNN regression predictions on 24 observations of real experiments. Note that the parameters are in the scale of log base $e$ . The left figure is for parameter $\gamma$ and the right figure is for parameter $\alpha$ .                                                | 128 |
| 3.23 | GNN regression ABC inference results on 24 observations of real experiments using the relative $\ell_1$ distance. Note that the parameters are in the scale of log base $e$ . The left figure is for parameter $\gamma$ and the right figure is for parameter $\alpha$ . | 129 |
| 3.24 | GNN regression ABC inference results on 24 observations of real experiments using the KL divergence. Note that the parameters are in the scale of log base $e$ . The left figure is for parameter $\gamma$ and the right figure is for parameter $\alpha$ .              | 130 |

|      |                                                                                                                                                                                                                                                                                                                                                                                                            |     |
|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 3.25 | Sampling validation result of GNN regression ABC inference. The left figure uses the relative distance as the distance function in ABC, and the right figure uses the KL divergence as the distance function in ABC. Note that the parameters are normalized. In each figure, the left parameter is $\gamma$ and the right parameter is $\alpha$ . . . . .                                                 | 131 |
| 3.26 | Experimental image of the last time point. This is from one of 24 experimental samples. As described in section 1.4, it was taken by my senior, Atiyeh Ahmadi [2]. . . . .                                                                                                                                                                                                                                 | 132 |
| 3.27 | Simulated images of the last time point. These 4 images are from 4 simulated samples given different parameter values: $\gamma = 1, \alpha = 50$ (top-left); $\gamma = 500, \alpha = 50$ (top-right); $\gamma = 500, \alpha = 0.5$ (bottom-left); $\gamma = 1, \alpha = 0.5$ (bottom-right). . . . .                                                                                                       | 132 |
| 3.28 | Cell radius parameter distribution comparison between simulations and experiments. The cell radius is one of the specified observable parameters in the <code>CellModeller</code> simulation module. . . . .                                                                                                                                                                                               | 134 |
| 3.29 | Division length threshold parameter distribution comparison between simulations and experiments. The division length threshold is one of the specified observable parameters in the <code>CellModeller</code> simulation module. . . . .                                                                                                                                                                   | 135 |
| 3.30 | Pickle step parameter distribution comparison between simulations and experiments. The pickle step parameter specifies the number of time steps between two records, where each time step is 0.025 seconds. The left figure shows the experimental distribution. The right figure shows both the experimental and simulated distributions when the random selection is applied to the pickle step. . . . . | 136 |
| 3.31 | Pickle step parameter distribution comparison between simulations and experiments. This simulated pickle step is specified as a deterministic value 140. . . . .                                                                                                                                                                                                                                           | 136 |
| B.1  | Posterior probability densities for calibrated parameter values of $\gamma$ and $\alpha$ after six rounds of ABC calibration [2]. . . . .                                                                                                                                                                                                                                                                  | 153 |
| B.2  | Global sensitivity analysis across $\gamma$ and $\alpha$ values. Red bars represent the standard deviation of each feature value determined from an ensemble of 50 simulations for each parameter pair [2]. . . . .                                                                                                                                                                                        | 154 |

|     |                                                                                                                                                                                                                                                                                               |     |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| B.3 | Summary statistics for experimental data and simulations sampled from prior distributions and final posterior distribution (n=24). Lower box boundary: 25th percentile; upper box boundary: 75th percentile; line inside box: median; whiskers: $1.5 \times$ interquartile range [2]. . . . . | 155 |
| B.4 | KL divergence between experimental data and simulations sampled from prior distributions and final posterior distributions shown in Figure B.3 [2].                                                                                                                                           | 155 |

# List of Tables

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |    |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 1.1 | CellModeller settings. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | 41 |
| 1.2 | ABC-SMC settings. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | 51 |
| 2.1 | Values of $\gamma$ and $\alpha$ . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | 64 |
| 2.2 | Transformations applied to the parameter $\theta$ during different stages for the regression tasks (i.e., the first and the second strategy) with relative distance of summary statistics. Note that the summary statistics may also require normalization if the relative distance is replaced with a distribution-based distance and the KL divergence is used. . . . .                                                                                                                                                                                                                                                                                                                                                                              | 78 |
| 2.3 | Transformations applied to the node features $X$ during different stages for the regression tasks (i.e., the first and the second strategy). Note that $X$ means the node feature matrix that includes each node feature $x$ . . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | 78 |
| 2.4 | Transformations applied to the summary statistics $S_\phi(x)$ during different stages for the regression tasks (i.e., the first and the second strategy). Note that the summary statistics may also require normalization if the relative distance is replaced with a distribution-based distance and the KL divergence is used. . . . .                                                                                                                                                                                                                                                                                                                                                                                                               | 79 |
| 2.5 | Transformations applied to the parameter $\theta$ during different stages for the embedding learning tasks (i.e., the third and the fourth strategy) with relative distance of summary statistics. The $g$ stands for the graph embedding, which aggregates all the node embeddings across the graph. The $h_i \in \mathbb{R}^d$ denotes the node embedding, and the $g \in \mathbb{R}^d$ denotes the graph embedding. Aggregation could be the mean graph pooling, which means $g = \frac{1}{ V } \sum_{i \in V} h_i$ . Note that the summary statistics require normalization, even when the relative distance is used. There is no standard score normalization for the node targets here because the triplets are applied as graph labels. . . . . | 88 |

|     |                                                                                                                                                                                                                                                                                                                                                                                                         |     |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 2.6 | Transformations applied to the node features $X$ during different stages for the embedding learning tasks (i.e., the third and the fourth strategy). Note that $X$ means the node feature matrix that includes each node feature $x$ . This table remains the same as in the first and second strategies. . . . .                                                                                       | 89  |
| 2.7 | Transformations applied to the summary statistics $S_\phi(x)$ during different stages for the embedding learning tasks (i.e., the third and the fourth strategy). The summary statistics here are graph representations aggregated by the graph pooling mechanism over all nodes in the graph. Note that the summary statistics require normalization, even when the relative distance is used. . . . . | 89  |
| 2.8 | Transformations applied to the parameter $\theta$ during different stages for the traditional approach (i.e., the fifth strategy) with relative distance of summary statistics. Note that $x_{\min} = \min(\log_{10}(\theta))$ and $x_{\max} = \max(\log_{10}(\theta))$ . . . . .                                                                                                                       | 94  |
| 2.9 | Transformations applied to the summary statistics $S(x)$ during different stages for the traditional approach (i.e., the fifth strategy). . . . .                                                                                                                                                                                                                                                       | 94  |
| 3.1 | Testing mean squared error (MSE) for different numbers of input features. . . . .                                                                                                                                                                                                                                                                                                                       | 114 |
| 3.2 | Simulation-based validation results for strategies when $\varepsilon = 0.02$ . . . . .                                                                                                                                                                                                                                                                                                                  | 120 |
| A.1 | Feature List with Data Types and Normalization . . . . .                                                                                                                                                                                                                                                                                                                                                | 150 |

# Chapter 1

## Introduction

### 1.1 Context Background

Systems biology seeks to define biological function as the result of many interacting components within a system [31]. It emphasizes the networked organization (structure) and its properties (nonlinear dynamics), as well as the functional regulation of these systems (feedback), to provide a context for examining how the interactions among biological components yield complex behaviors.

Mechanistic modeling enables the quantitative characterization of biological systems. Expressing hypotheses about the structure and dynamics of a biological system in quantitative terms (e.g., mathematical equations, systematic analysis of stability, robustness, and parameter sensitivity) can help explore the behavior of regions or regimes that may be difficult or impractical to investigate experimentally [33]. The growth of microbial populations is a common mechanistic model in systems biology.

Mechanistic modeling plays a key role in systems biology. For such models to generate accurate predictions, they must be calibrated against experimental observations. This calibration task, sometimes referred to as model fitting, involves estimating the values of model parameters, typically by selecting those that best align the model's behavior with observations.

When those observations involve the spatial distribution of system components (cells, molecules, tissues, and organisms), the alignment may require comparing model simulations with imaging data (e.g., end-point or time-lapse). When model output and observed images are not directly aligned, or when variability implies that observations are samples from a

random process, the goal of aligning behavior becomes aligning 'patterns' or 'trends' that represent the underlying behavior. In the ecology literature, this has been codified as 'Pattern-Oriented Modeling'. Calibration can be MLE-based (point estimate) or, more usefully, capture uncertainty (local linear or Bayesian) [22].

## 1.2 Agent-based Model

### 1.2.1 Definition

The Agent-Based Model (ABM) is a mathematical and computational framework used to model systems composed of many interacting individual entities (agents) [43]. Instead of modeling the system globally, ABMs describe the behavior of each agent and study the emergent collective dynamics.

Consider  $N$  agents. Each agent  $i$  has a state

$$x_i(t) \in \mathcal{S}, \quad (1.1)$$

where  $\mathcal{S}$  is the state space.

The full system state is

$$X(t) = (x_1(t), x_2(t), \dots, x_N(t)) \in \mathcal{S}^N. \quad (1.2)$$

Agents interact through a graph

$$G = (V, E), \quad (1.3)$$

where  $V = \{1, \dots, N\}$  and  $E$  is the set of edges.

The neighborhood of agent  $i$  is

$$\mathcal{N}_i(t) = \{j \in V : (i, j) \in E(t)\}. \quad (1.4)$$

Each agent evolves according to

$$x_i(t+1) = F_i(x_i(t), \mathcal{N}_i(t), E(t), \xi_i(t)), \quad (1.5)$$

where  $F_i$  is the update function;  $\mathcal{N}_i(t)$  is the set of neighbors;  $E(t)$  represents the environment;  $\xi_i(t)$  is an i.i.d. random variable (noise).

ABMs are stochastic and can be described by the transition probability,

$$P_{ij} = \mathbb{P}(X(t+1) = j \mid X(t) = i), \quad (1.6)$$

where the state of  $t+1$  depends on the state of  $t$  only.

The matrix

$$P = (P_{ij})_{i,j \in \mathcal{S}} \quad (1.7)$$

is called the transition matrix, satisfying

$$P_{ij} \geq 0, \quad \sum_j P_{ij} = 1. \quad (1.8)$$

### 1.2.2 Advantages of Simulation

There are several properties of ABM that are the reasons why it was selected for this project [42].

1. Heterogeneity of individuals.

Classical deterministic models, such as ODEs, assume homogeneous populations are

$$\frac{dx}{dt} = f(x), \quad (1.9)$$

where  $x(t)$  represents an aggregate state.

However, in many systems, individuals differ significantly. In ABMs, each agent  $i$  has its own state

$$x_i(t) \in \mathcal{S}, \quad i = 1, \dots, N. \quad (1.10)$$

So, as Equation 1.5 described, for each cell  $i$ , it has its own  $F_i$ .

2. Local interactions and emergence.

Also as Equation 1.5 described,  $x_i(t+1)$  updates based on its local neighbors  $\mathcal{N}_i(t)$ , so Equation 1.5 can be written as

$$x_i(t+1) = F_i(x_i(t), \{X_j(t) : j \in \mathcal{N}_i(t)\}, E(t), \xi_i(t)). \quad (1.11)$$

Macroscopic patterns emerge from microscopic rules, which cannot be easily captured by global deterministic equations.

3. Spatial structure and discreteness.

The agent state update function  $F_i$  for an arbitrary agent  $i$  Equation 1.5 can be written as the spatial movement function as

$$x_i(t+1) = x_i(t) + v_i(t), \quad (1.12)$$

where each agent  $i$  has a spatial position

$$x_i(t) \in \mathbb{R}^d, \quad (1.13)$$

at time point  $t$

ABMs are good at capturing such spatial structure and dynamics, especially when the time is not continuous.

4. Intrinsic stochasticity.

Still, as Equation 1.5 described, for any arbitrary agent  $i$ , the update function  $F_i$  is in terms of the i.i.d. noise variable  $\xi_i(t)$  to naturally incorporate randomness. This makes the dynamical system evolve as a Markov process over time.

5. Complex and nonlinear rules.

As described in Equation 1.6, the transition probability is applied to update every agent state  $x_i$  where  $i = 1, \dots, N$ , when ABM systems involve rules that cannot be expressed as closed-form equations, so ABMs directly simulate the system state

$$X(t+1) \sim P_{ij} = \mathbb{P}(X(t+1) = j \mid X(t) = i), \quad (1.14)$$

based on nonlinear rules  $F_i$  for all the  $i = 1, \dots, N$ , and are called agencies for each agent  $i$ .

6. Flexibility and modularity.

As Equation 1.5 described, ABMs allow easy modification of rules

$$X_i(t+1) = F_i(\cdot). \quad (1.15)$$

Changing behavior only requires redefining  $F_i$ , unlike ODE/PDE models, which may require complete reformulation. Thus, ABMs can be easily customized in different ways.

### 1.2.3 Difficulties of Calibration

There are several difficulties that we are facing in calibrating ABMs. Some of the following difficulties of calibration are even the same as the advantages of simulation for ABMs [62].

1. Issue with the noise of the experimental data.

Providing noise does not affect the correctness of calibration within a specified credible range, but the calibration result would be meaningless because the correctness arises from a large variance in the posterior distribution. This is a cheating way to more likely cover the true parameter by adjusting the bias-variance trade-off.

2. Issue with maximum likelihood estimation.

Calibration refers to estimating model parameters  $\theta$  such that the model output matches observed data.

The goal is to perform maximum likelihood estimation

$$\hat{\theta} = \arg \max_{\theta} p(Y | \theta), \quad (1.16)$$

where  $Y$  is the observed data;  $\theta$  is the parameter vector;  $p(Y | \theta)$  is the likelihood of observing  $Y$  under parameters  $\theta$ .

If we define  $Y$  as  $Y_{sim}$  which means the simulated data, then

$$Y_{sim} = g(X(0), X(1), \dots, X(N)), \quad (1.17)$$

where  $g(\cdot)$  is the observation or summary function, and it maps the full trajectory of the system to observable outputs. Here,  $Y$  indicates the observed data, which can also be written as  $Y_{obs}$  to distinguish from  $Y_{sim}$ .

The objective is to find the parameter  $\hat{\theta}$  that makes the observed data most probable.

In many standard models, the likelihood can be written in closed form. However, in ABMs, the model is defined through simulation rather than an explicit probability density, making this task significantly more difficult.

3. Issue with intractable likelihood function.

As described in [Equation 1.6](#), an ABM can be represented as a stochastic process

$$X(t+1) \sim P_{\theta}(X(t)), \quad (1.18)$$

where  $X(t)$  is the system state at time  $t$ ;  $P_\theta$  is the transition probability depending on parameters  $\theta$ .

Suppose we observe data  $Y$ . The likelihood can be expressed as

$$p(Y | \theta) = \int p(Y, X | \theta) dX, \quad (1.19)$$

by the law of total probability, and we have

$$p(Y | \theta) = \int p(Y | X, \theta) p(X | \theta) dX, \quad (1.20)$$

by the chain rule, and we know

$$Y \perp \theta | X \Rightarrow p(Y | X, \theta) = p(Y | X). \quad (1.21)$$

So that we have

$$p(Y | \theta) = \int p(Y | X(\theta)) p(X | \theta) dX, \quad (1.22)$$

or to make it simpler, we write it as

$$p(Y | \theta) = \int p(Y | X) p_\theta(X) dX, \quad (1.23)$$

where  $X$  represents latent system trajectories;  $p_\theta(X)$  is the probability of a trajectory under parameters  $\theta$ ;  $p(Y | X)$  is the observation model.

Thus, computing the likelihood requires integrating over all possible hidden trajectories  $X$ . The likelihood  $p(Y | \theta)$  is typically not available in closed form and cannot be computed exactly.

To handle this issue, some computational inference methods can be applied, such as the approximate Bayesian computation method, and more details will be provided in the later sections.

#### 4. Models are always high-dimensional.

As described in [Equation 1.2](#), the system state is

$$X(t) = (X_1(t), \dots, X_N(t)) \in \mathcal{S}^N. \quad (1.24)$$

The size of the state space grows exponentially. That is

$$|\mathcal{S}^N| = |\mathcal{S}|^N. \quad (1.25)$$

This leads to the curse of dimensionality and the impossibility of enumerating or integrating over all states.

5. Issue with the noise of the simulated data.

Even with fixed parameters  $\theta$ , the model output is random. That is

$$Y^{(k)} \sim p(Y \mid \theta), \quad k = 1, 2, \dots \quad (1.26)$$

This means the parameter  $\theta$  is fixed, the ABM is run multiple times, and each run produces a different output  $Y^{(k)}$ . That means, if there are  $K$  runs given the fixed  $\theta$ , any arbitrary  $Y^{(i)}$  and  $Y^{(j)}$  can be very different where  $i \neq j$  and  $i, j = 1, \dots, K$ .

Another way to say it is that it causes non-identifiability, since different parameter values can fit the same data equally well. That means

$$p(Y \mid \theta_1) \approx p(Y \mid \theta_2), \quad \theta_1 \neq \theta_2. \quad (1.27)$$

This randomness arises from stochastic events within the simulation.

In calibration, we often minimize a loss function such as

$$L(\theta) = \|Y_{\text{obs}} - Y_{\text{sim}}(\theta)\|. \quad (1.28)$$

However, since  $Y_{\text{sim}}(\theta)$  is random, the loss function itself is noisy.

Thus, the optimizer does not observe a stable objective function, but rather a noisy one.

To reduce variability, we estimate the expected output as

$$\mathbb{E}[Y \mid \theta] \approx \frac{1}{K} \sum_{k=1}^K Y^{(k)}. \quad (1.29)$$

This implies that the true quantity of interest is  $\mathbb{E}[Y \mid \theta]$ , and it is approximated using  $K$  independent simulation runs.

While averaging reduces randomness, it introduces Monte Carlo error.

That says, there is a trade-off between accuracy and computational cost. A small  $K$  means low computational cost but high variance. A large  $K$  means lower variance but high computational cost.

Gradient-based optimization methods assume a smooth objective function.

However, due to simulation noise, the gradients may be unstable or inaccurate, and the optimization may become unreliable or fail.

Thus, stochastic simulation output significantly complicates parameter calibration in ABMs.

6. Simulations are computationally expensive.

Each evaluation of the likelihood  $p(Y \mid \theta)$  requires running the full simulation. Let  $T$  denote the simulation horizon, and  $N$  denote the number of agents.

Then, the computational cost of a single simulation is approximately

$$\mathcal{O}(N \cdot T). \quad (1.30)$$

This scaling reflects that the cost grows with both the number of agents and the length of the simulation.

Calibration typically requires  $K$  simulation replications per parameter value, and  $M$  parameter values to be evaluated.

Thus, the total computational cost is

$$\mathcal{O}(K \cdot M \cdot N \cdot T), \quad (1.31)$$

where  $N$  says that more agents imply more computations per time step;  $T$  says that longer simulations require more time steps;  $M$  indicates the number of parameter candidates explored during optimization;  $K$  indicates the number of repeated simulations needed to reduce stochastic noise.

Therefore, calibration cost scales multiplicatively with all these factors, making it computationally expensive.

## 1.3 Deep Learning Preliminaries

### 1.3.1 Neural Network and Basics

A neural network is a function that maps an input vector to an output through a sequence of linear transformations and nonlinear activation functions [21].

A neuron is simply a nonlinear function of a weighted sum. Mathematically,

$$y = \sigma(w^T x + b), \quad (1.32)$$

where  $x \in \mathbb{R}^d$  indicates the input vector;  $w \in \mathbb{R}^d$  indicates the weight vector;  $b \in \mathbb{R}$  indicates the bias;  $\sigma(\cdot)$  indicates the activation function;  $y$  indicates the output;  $d$  indicates the size of the input vector.

A layer comprises multiple neurons. Instead of one weight vector, we use a weight matrix. Mathematically,

$$y = \sigma(W^T x + b), \quad (1.33)$$

where  $m$  indicates the number of neurons;  $W \in \mathbb{R}^{m \times d}$  indicates the weight matrix;  $b \in \mathbb{R}^m$  indicates the bias.

A network comprises multiple layers. Let the input be

$$a^{(0)} = x. \quad (1.34)$$

For each layer  $l = 1, 2, \dots, k$ , the network computes two steps:

1. Linear transformation:

$$z^{(l)} = W_l a^{(l-1)} + b_l, \quad (1.35)$$

where  $W_l$  is the weight matrix of layer  $l$ ;  $b_l$  is the bias vector.

2. Activation function:

$$a^{(l)} = \sigma(z^{(l)}), \quad (1.36)$$

where  $\sigma(\cdot)$  is the activation function.

After the last layer  $k$ , the network output is

$$f(x) = a^{(k)}. \quad (1.37)$$

Equivalently,

$$f(x) = \sigma(W_k a^{(k-1)} + b_k). \quad (1.38)$$

If we ignore biases for simplicity, the neural network can be written as

$$f(x) = W_k \sigma(W_{k-1} \dots \sigma(W_1 x)). \quad (1.39)$$

The network prediction is compared to the true value using the loss function. A common example is Mean Squared Error (MSE), which is mathematically defined as

$$L = \frac{1}{N} (\hat{y} - y)^2, \quad (1.40)$$

where  $\hat{y}$  indicates the network prediction;  $y$  indicates the true value which causes a zero loss;  $N$  indicates the number of samples.

To lower the MSE, the goal is

$$\min_{\theta} L = \min_{\theta} \frac{1}{N} (\hat{y} - y)^2, \quad (1.41)$$

where  $\theta$  indicates  $\{W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, \dots, W^{(k)}, b^{(k)}\}$ , the set of parameters (weights and biases) the model learns for a  $k$ -layer neural network.

Forward propagation is used to predict on the input data and compute the loss for the current batch. Backpropagation is used to update the weights using a learning step and the gradient for the next batch [49]. Each batch consists of one forward and one backward pass. Each epoch trains the model once on all the training data, which can be split into batches.

To update weights and biases, we need the gradients

$$\frac{\partial L}{\partial W^{(1)}}, \frac{\partial L}{\partial b^{(1)}}, \quad (1.42)$$

since the loss actually depends on the weights  $L(W^{(1)}, W^{(2)}, \dots)$  and biases  $L(b^{(1)}, b^{(2)}, \dots)$ .

For example, for the first layer weight matrix  $W^{(1)}$ , the update rule is

$$W^{(1)} \leftarrow W^{(1)} - \eta \frac{\partial L}{\partial W^{(1)}}. \quad (1.43)$$

Similarly, the bias update is

$$b^{(1)} \leftarrow b^{(1)} - \eta \frac{\partial L}{\partial b^{(1)}}, \quad (1.44)$$

where  $\eta$  = learning rate.

The gradient specifies the direction of parameter updates for the current batch, and the learning rate specifies the magnitude of the changes.

To briefly introduce how backpropagation mathematically works in neural networks, we need to determine the gradients

$$\frac{\partial L}{\partial W^{(l)}}, \frac{\partial L}{\partial b^{(l)}}, \quad (1.45)$$

where  $l = 1, 2, 3, \dots, k$  indicates the layer index; the chain rule is applied as follows.

For each layer  $l$ , define the error term as

$$\delta^{(l)} = \frac{\partial L}{\partial z^{(l)}}, \quad (1.46)$$

where  $L$  denotes the loss function and  $z^{(l)}$  is the pre-activation vector of layer  $l$ . The quantity  $\delta^{(l)}$  measures the sensitivity of the loss with respect to the pre-activation values at layer  $l$ .

Since  $L$  depends on  $W^{(l)}$  through  $z^{(l)}$ , apply chain rule

$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial z^{(l)}} \cdot \frac{\partial z^{(l)}}{\partial W^{(l)}}. \quad (1.47)$$

So,

$$\frac{\partial L}{\partial W^{(l)}} = \delta^{(l)} \cdot \frac{\partial z^{(l)}}{\partial W^{(l)}}. \quad (1.48)$$

And as described in [Equation 1.35](#), we know

$$\frac{\partial z^{(l)}}{\partial W^{(l)}} = \frac{\partial (W^{(l)} a^{(l-1)} + b^{(l)})}{\partial W^{(l)}} = a^{(l-1)}. \quad (1.49)$$

The forward propagation equation is

$$z^{(l+1)} = W^{(l+1)} a^{(l)} + b^{(l+1)}, \quad (1.50)$$

$$a^{(l)} = \sigma(z^{(l)}), \quad (1.51)$$

where  $W^{(l+1)}$  and  $b^{(l+1)}$  denote the weights and biases of layer  $l+1$ , and  $\sigma(\cdot)$  is the activation function.

Thus, the dependency between variables can be written as

$$z^{(l)} \rightarrow a^{(l)} \rightarrow z^{(l+1)} \rightarrow L. \quad (1.52)$$

To compute the derivative of the loss with respect to  $z^{(l)}$ , we apply the chain rule as

$$\frac{\partial L}{\partial z^{(l)}} = \frac{\partial L}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial z^{(l)}}. \quad (1.53)$$

In this equation:

1. By definition of the error term, we know

$$\delta^{(l+1)} = \frac{\partial L}{\partial z^{(l+1)}}. \quad (1.54)$$

2. From the linear transformation, we know

$$z^{(l+1)} = W^{(l+1)}a^{(l)} + b^{(l+1)}; \quad (1.55)$$

So we obtain

$$\frac{\partial z^{(l+1)}}{\partial a^{(l)}} = W^{(l+1)}. \quad (1.56)$$

3. Since

$$a^{(l)} = \sigma(z^{(l)}). \quad (1.57)$$

We have

$$\frac{\partial a^{(l)}}{\partial z^{(l)}} = \sigma'(z^{(l)}), \quad (1.58)$$

where the derivative is applied elementwise.

Combining the above three expressions yields the recursive backpropagation formula

$$\delta^{(l)} = (W^{(l+1)})^T \delta^{(l+1)} \odot \sigma'(z^{(l)}), \quad (1.59)$$

where  $\odot$  denotes elementwise multiplication [49].

Once  $\delta^{(l)}$  is computed, the gradient of the loss with respect to the weight matrix of layer  $l$  is given by

$$\frac{\partial L}{\partial W^{(l)}} = \delta^{(l)}(a^{(l-1)})^T = [(W^{(l+1)})^T \delta^{(l+1)} \odot \sigma'(z^{(l)})](a^{(l-1)})^T, \quad (1.60)$$

where the same way is applied to get the formulation of  $\frac{\partial L}{\partial b^{(l)}}$  described in Equation 1.45.

In addition, training is terminated when the validation loss does not improve for a predefined number of consecutive epochs, referred to as *patience*. This procedure prevents unnecessary training after the model begins to overfit the training data and helps maintain better generalization performance. That being said, let  $L_{\text{val}}^{(t)}$  denote the validation loss in epoch  $t$ . The optimal model parameters are selected according to

$$\theta^* = \arg \min_t L_{\text{val}}^{(t)}. \quad (1.61)$$

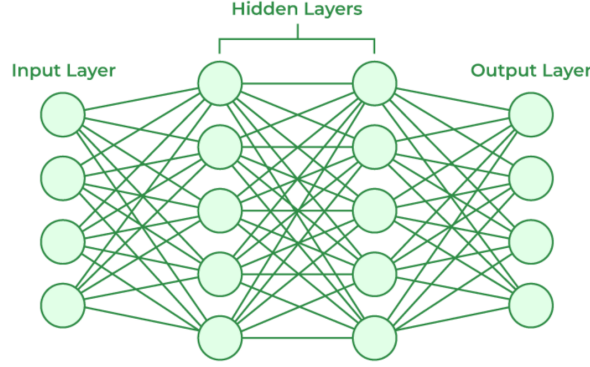


Figure 1.1: Neural Network Architecture. This figure is obtained from <https://www.geeksforgeeks.org/deep-learning/artificial-neural-networks-and-its-applications/>.

### 1.3.2 Recurrent Neural Network and Long Short-term Memory

#### Recurrent Neural Network

The Recurrent Neural Network (RNN) architecture is a network designed to process sequential, time-ordered data, such as time series, text, or periodic signals (e.g., sounds or vibrations). RNNs implement a feedback loop that allows them to use information from the previous time step when making predictions. At each time step, RNNs adjust to reflect the input and changes to form the hidden state. Thus, RNNs compute a hidden layer that reflects the current input and the previous hidden layer, and ultimately generate an output using the hidden-layer values at each time step. [16].

Suppose that sequential samples are given as

$$x_1, x_2, \dots, x_T, \quad (1.62)$$

where  $x_t$  denotes the input at time step  $t$ .

At time step  $t$ , the hidden state  $h_t$  is computed as

$$h_t = \sigma(W_x x_t + W_h h_{t-1} + b), \quad (1.63)$$

where  $x_t$  denotes the input vector at time  $t$ ,  $h_t$  represents the hidden state, and  $h_{t-1}$  is the hidden state from the previous time step. The matrices  $W_x$  and  $W_h$  are the input and recurrent weight matrices that are shared over time  $t$ , respectively,  $b$  is a bias vector, and  $\sigma(\cdot)$  is an activation function such as tanh or ReLU.

Note that the input weights  $W_x$  connect the input  $x_t$  to the hidden state, and the recurrent weights  $W_h$  connect the previous hidden state  $h_{t-1}$  to the new hidden state.

If the network produces an output at each time step, the output prediction can be computed as

$$y_t = g(W_y h_t + c), \quad (1.64)$$

for  $t = 1, 2, \dots, T$ , where  $T$  denotes the length of the sequence.

The initial hidden state is typically initialized as

$$h_0 = 0, \quad (1.65)$$

where  $W_y$  is the output weight matrix,  $c$  is a bias vector, and  $g(\cdot)$  is an output activation function such as the softmax or sigmoid function.

## Vanishing and Exploding Gradients

We used LSTMs instead of RNNs to mitigate vanishing and exploding gradients.

To introduce the problem of vanishing and exploding gradients in RNNs, we suppose the loss depends on the final hidden state of RNNs

$$L = L(h_T). \quad (1.66)$$

Let us take  $W_h$  as an example here.  $W_x$  can be explained in the same way. We know  $W_h$  affects  $h_1, h_2, \dots, h_T$ , and  $h_T$  is a function of  $h_1, h_2, \dots, h_{T-1}$ , so  $L(h_T)$  depends on  $h_1, h_2, \dots, h_{T-1}, h_T$ .

Then, by the multivariable chain rule, the gradient of the recurrent weight is

$$\frac{\partial L}{\partial W_h} = \sum_{t=1}^T \frac{\partial L}{\partial h_t} \frac{\partial h_t}{\partial W_h}, \quad (1.67)$$

for  $t = 1, 2, \dots, T$ , where  $T$  denotes the length of the sequence. Note that  $W_h$  is shared at all times  $t$  as described in [Equation 1.63](#).

So we need

$$\frac{\partial L}{\partial h_t}. \quad (1.68)$$

Using the chain rule,

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_T} \frac{\partial h_T}{\partial h_t}. \quad (1.69)$$

Thus, the key quantity to analyze is

$$\frac{\partial h_T}{\partial h_t}. \quad (1.70)$$

Using the chain rule repeatedly,

$$\frac{\partial h_T}{\partial h_t} = \frac{\partial h_T}{\partial h_{T-1}} \frac{\partial h_{T-1}}{\partial h_{T-2}} \dots \frac{\partial h_{t+1}}{\partial h_t}. \quad (1.71)$$

Each term is a

$$\frac{\partial h_k}{\partial h_{k-1}} = \text{diag}(\sigma'(z_k)) W_h. \quad (1.72)$$

It is derived in the following steps:

1. The basic recurrent neural network is described as [Equation 1.63](#), so the hidden state is computed as

$$h_k = \sigma(z_k), \quad (1.73)$$

where

$$z_k = W_h h_{k-1} + W_x x_k + b. \quad (1.74)$$

Thus, the computation can be viewed as a two-step process. That is

$$h_{k-1} \rightarrow z_k \rightarrow h_k. \quad (1.75)$$

2. We aim to compute the derivative

$$\frac{\partial h_k}{\partial h_{k-1}}. \quad (1.76)$$

Using the chain rule for vector-valued functions, we obtain

$$\frac{\partial h_k}{\partial h_{k-1}} = \frac{\partial h_k}{\partial z_k} \frac{\partial z_k}{\partial h_{k-1}}. \quad (1.77)$$

Thus, the derivative can be computed by evaluating the two Jacobian matrices separately.

3. From Equation 1.74, only the first term depends on  $h_{k-1}$ .

Therefore,

$$\frac{\partial z_k}{\partial h_{k-1}} = W_h. \quad (1.78)$$

4. Since

$$h_k = \sigma(z_k), \quad (1.79)$$

and the activation is applied elementwise,

$$h_{k,i} = \sigma(z_{k,i}), \quad (1.80)$$

we compute the Jacobian

$$\frac{\partial h_{k,i}}{\partial z_{k,j}}. \quad (1.81)$$

When  $i = j$ ,

$$\frac{\partial h_{k,i}}{\partial z_{k,i}} = \sigma'(z_{k,i}). \quad (1.82)$$

When  $i \neq j$ ,

$$\frac{\partial h_{k,i}}{\partial z_{k,j}} = 0. \quad (1.83)$$

Thus, the Jacobian is diagonal.

$$\frac{\partial h_k}{\partial z_k} = \text{diag}(\sigma'(z_k)) = \begin{pmatrix} \sigma'(z_{k,1}) & 0 & \cdots \\ 0 & \sigma'(z_{k,2}) & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}. \quad (1.84)$$

This is a diagonal matrix because each component  $h_{k,i}$  depends only on the corresponding input  $z_{k,i}$ .

5. Substituting the two Jacobians into the chain rule yields

$$\frac{\partial h_k}{\partial h_{k-1}} = \text{diag}(\sigma'(z_k)) \frac{\partial z_k}{\partial h_{k-1}}. \quad (1.85)$$

Since

$$\frac{\partial z_k}{\partial h_{k-1}} = W_h, \quad (1.86)$$

we obtain

$$\frac{\partial h_k}{\partial h_{k-1}} = \text{diag}(\sigma'(z_k)) W_h. \quad (1.87)$$

Thus, applying the chain rule across multiple time steps yields

$$\frac{\partial h_T}{\partial h_t} = \prod_{k=t+1}^T \text{diag}(\sigma'(z_k)) W_h. \quad (1.88)$$

This means that the gradient is a product of many matrices, so it is not stable.

To analyze whether gradients grow or decay through time, we consider the norm of the Jacobian

$$\left\| \frac{\partial h_T}{\partial h_t} \right\|. \quad (1.89)$$

Using the submultiplicative property of matrix norms, we know for matrix A and B,

$$\|AB\| \leq \|A\| \|B\|. \quad (1.90)$$

So, we obtain

$$\left\| \frac{\partial h_T}{\partial h_t} \right\| \leq \prod_{k=t+1}^T \|\text{diag}(\sigma'(z_k)) W_h\|. \quad (1.91)$$

Using the submultiplicative property of matrix norms again, we know for matrix  $\text{diag}(\sigma'(z_k))$  and  $W_h$ ,

$$\|\text{diag}(\sigma'(z_k)) W_h\| \leq \|\text{diag}(\sigma'(z_k))\| \|W_h\|. \quad (1.92)$$

So, we obtain

$$\left\| \frac{\partial h_T}{\partial h_t} \right\| \leq \prod_{k=t+1}^T \|\text{diag}(\sigma'(z_k))\| \|W_h\|. \quad (1.93)$$

Define

$$\gamma = \sup_z |\sigma'(z)|. \quad (1.94)$$

This quantity represents the largest possible magnitude of the derivative of the activation function.

Because

$$|\sigma'(z_k)| \leq \gamma. \quad (1.95)$$

That means

$$|\sigma'(z_{k,i})| \leq \gamma \quad \text{for all } i. \quad (1.96)$$

So,

$$\max_i |\sigma'(z_{k,i})| \leq \gamma. \quad (1.97)$$

And we know the norm of a diagonal matrix is controlled by the largest absolute diagonal entry

$$\|\text{diag}(\sigma'(z_k))\| = \max_i |\sigma'(z_{k,i})|. \quad (1.98)$$

Then we obtain

$$\|\text{diag}(\sigma'(z_k))\| \leq \gamma. \quad (1.99)$$

For common activation functions, the derivative is bounded. For example,

$$\sigma'(z) \leq 0.25 \quad (\text{sigmoid}), \quad (1.100)$$

$$|\sigma'(z)| \leq 1 \quad (\text{tanh}). \quad (1.101)$$

Therefore,

$$\gamma \leq 1. \quad (1.102)$$

Substituting the above bound gives

$$\left\| \frac{\partial h_T}{\partial h_t} \right\| \leq (\gamma \|W_h\|)^{T-t}. \quad (1.103)$$

This inequality shows that the magnitude of the gradient depends exponentially on the sequence length  $T - t$ .

If

$$\gamma \|W_h\| < 1, \quad (1.104)$$

then

$$(\gamma \|W_h\|)^{T-t} \rightarrow 0, \quad (1.105)$$

as  $T - t$  increases, which leads to the *vanishing gradient problem*.

Conversely, if

$$\gamma \|W_h\| > 1, \quad (1.106)$$

then

$$(\gamma \|W_h\|)^{T-t} \rightarrow \infty, \quad (1.107)$$

as  $T - t$  increases, which results in the *exploding gradient problem*.

## Long Short-term Memory

Long Short-Term Memory (LSTM) networks are a specialized form of RNNs designed to mitigate vanishing and exploding gradients. The LSTM network maintains two states at each time step: the hidden state  $h_t$  and the cell state  $c_t$ . The hidden state represents the output of the network, while the cell state serves as a memory that carries long-term information across time steps [26].

At time step  $t$ , the LSTM computes the following quantities.

1. Forget gate

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f). \quad (1.108)$$

2. Input gate

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i). \quad (1.109)$$

3. Candidate memory

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c). \quad (1.110)$$

4. Cell state update

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t. \quad (1.111)$$

where input gate  $f_t$  controls how much old information  $c_{t-1}$  to keep, and input gate  $i_t$  controls how much new information  $\tilde{c}_t$  to write.

5. Output gate

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o), \quad (1.112)$$

where this controls which part of the aggregated information is the output.

6. Hidden state

$$h_t = o_t \odot \tanh(c_t). \quad (1.113)$$

Note that  $x_t$  indicates the input vector at time step  $t$ ;  $W_f, W_i, W_c, W_o$  indicates the input weight matrices of four gates respectively;  $U_f, U_i, U_c, U_o$  indicates the recurrent weight matrices of four gates respectively;  $b_f, b_i, b_c, b_o$  indicates the bias vectors of four gates respectively;  $\sigma(\cdot)$  indicates the sigmoid activation function;  $\tanh(\cdot)$  indicates the hyperbolic tangent activation;  $\odot$  indicates the elementwise product.

Suppose

$$x_t \in \mathbb{R}^d, \quad h_t, c_t \in \mathbb{R}^n. \quad (1.114)$$

For any of the input weight matrices, recurrent weight matrices or bias vectors associated with the gates,

$$W_* \in \mathbb{R}^{n \times d}, \quad U_* \in \mathbb{R}^{n \times n}, \quad b_* \in \mathbb{R}^n. \quad (1.115)$$

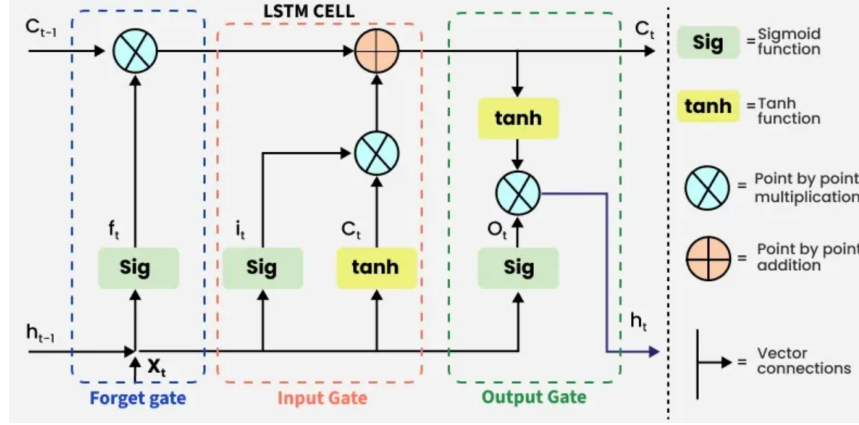


Figure 1.2: LSTM cell architecture <https://www.geeksforgeeks.org/deep-learning/deep-learning-introduction-to-long-short-term-memory/>.

## Mitigation of the Vanishing and Exploding Gradient Problem

LSTM addresses the vanishing and exploding gradient problems that RNNs could not solve [26].

As described in Equation 1.111, the cell state update is

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t. \quad (1.116)$$

Treat  $f_t, i_t, \tilde{c}_t$  as fixed with respect to  $c_{t-1}$ . Then

$$\frac{\partial c_t}{\partial c_{t-1}} = \text{diag}(f_t). \quad (1.117)$$

Or

$$c_{t,j} = f_{t,j} c_{t-1,j} + i_{t,j} \tilde{c}_{t,j}. \quad (1.118)$$

So

$$\frac{\partial c_{t,j}}{\partial c_{t-1,j}} = f_{t,j}, \quad \frac{\partial c_{t,j}}{\partial c_{t-1,\ell}} = 0 \quad \text{for } j \neq \ell. \quad (1.119)$$

Hence

$$\frac{\partial c_t}{\partial c_{t-1}} = \text{diag}(f_t). \quad (1.120)$$

Applying the chain rule over multiple time steps

$$\frac{\partial c_T}{\partial c_t} = \frac{\partial c_T}{\partial c_{T-1}} \frac{\partial c_{T-1}}{\partial c_{T-2}} \cdots \frac{\partial c_{t+1}}{\partial c_t}. \quad (1.121)$$

So

$$\frac{\partial c_T}{\partial c_t} = \prod_{k=t+1}^T \text{diag}(f_k). \quad (1.122)$$

Using a compatible matrix norm

$$\left\| \frac{\partial c_T}{\partial c_t} \right\| \leq \prod_{k=t+1}^T \|\text{diag}(f_k)\|. \quad (1.123)$$

Since  $f_k \in (0, 1)$  based on the activation function, we have

$$\|\text{diag}(f_k)\| \leq 1. \quad (1.124)$$

Therefore

$$\left\| \frac{\partial c_T}{\partial c_t} \right\| \leq 1. \quad (1.125)$$

Because  $f_t \in (0, 1)$ , the following product can still shrink.

$$\prod_{k=t+1}^T f_k. \quad (1.126)$$

So gradients can still vanish if many forget gates are much smaller than 1.

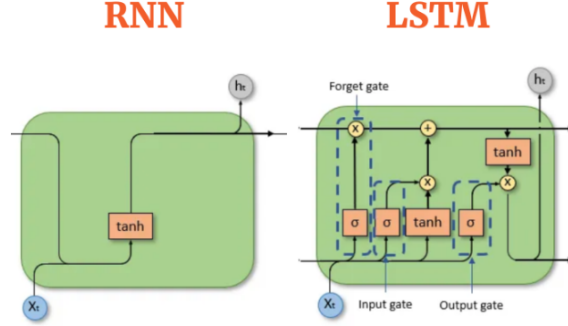


Figure 1.3: Comparing different Sequence models: RNN, LSTM. This figure is obtained from <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>.

### 1.3.3 Graph Neural Network and Multi-relational Graph

Graph Neural Networks (GNNs) are explicitly constructed to work with graph-based data structures, while most traditional deep learning architectures assume that the data are regular-shaped datasets (e.g., images or sequences) [51]. Regular-shaped datasets are assumed to have fixed data structures, so the input layer of traditional neural networks (e.g., fully connected or convolutional neural networks) takes the same input size regardless of the internal data structure [40]. GNNs are applied to this project because they do not rely on regular-shaped datasets and because their input layers take graph-based data, which have different sizes in different cases. Moreover, GNNs do not need to encode physical distances from the image data, whereas CNNs do.

GNN architectures treat the connectivity between two objects (e.g., an edge) as a first-order relationship. This architecture also makes GNNs suitable for representing systems in which interaction among various objects is critical to their overall behavior [20]. Sometimes, it is really difficult to define the meaning of graph-based data, but formally, a graph,

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}), \quad (1.127)$$

is defined by a set of nodes  $\mathcal{V}$  and a set of edges  $\mathcal{E}$  between these nodes. We denote an edge going from node  $u \in \mathcal{V}$  to node  $v \in \mathcal{V}$  as  $(u, v) \in \mathcal{E}$ . In many cases, we will be concerned only with *simple graphs*, where there is at most one edge between each pair of nodes, no edges between a node and itself, and where the edges are all undirected, i.e.,

$$(u, v) \in \mathcal{E} \iff (v, u) \in \mathcal{E}. \quad (1.128)$$

An easier way to represent graphs is to define an *adjacency matrix* as

$$\mathbf{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}. \quad (1.129)$$

Then represent the presence of edges as entries in this matrix:

$$\mathbf{A}[u, v] = \begin{cases} 1, & (u, v) \in \mathcal{E}, \\ 0, & \text{otherwise.} \end{cases} \quad (1.130)$$

If the graph contains only undirected edges, then  $\mathbf{A}$  is a symmetric matrix. However, if the graph is directed (i.e., edge direction matters), then  $\mathbf{A}$  is not necessarily symmetric. Some graphs may also have *weighted edges*, in which case the entries in the adjacency matrix are arbitrary real values rather than  $\{0, 1\}$ . For example, in a protein–protein interaction graph, edge weights may indicate the strength of association between two proteins [23].

In graph-based data, GNNs encounter many types of edges. Undirected, directed, and weighted edges are not the only ways to distinguish types of edges. We extend the edge notation to include a relation type  $\tau$ , i.e.,

$$(u, \tau, v) \in \mathcal{E}, \quad (1.131)$$

and define one adjacency matrix  $\mathbf{A}_\tau$  per edge type. These graphs are referred to as *multi-relational graphs*, and the entire graph can be represented by an adjacency tensor

$$\mathcal{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{R}| \times |\mathcal{V}|}, \quad (1.132)$$

where  $\mathcal{R}$  denotes the set of relation types. Two important subclasses of multi-relational graphs are *heterogeneous graphs* and *multiplex graphs*.

In heterogeneous graphs, nodes are associated with types, allowing the node set to be partitioned into disjoint subsets

$$\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2 \cup \dots \cup \mathcal{V}_k, \quad \text{with } \mathcal{V}_i \cap \mathcal{V}_j = \emptyset \text{ for } i \neq j. \quad (1.133)$$

Edges in heterogeneous graphs typically obey constraints induced by node types. A common constraint is that edges only connect nodes of specific types, i.e.,

$$(u, \tau_i, v) \in \mathcal{E} \Rightarrow u \in \mathcal{V}_j, v \in \mathcal{V}_k. \quad (1.134)$$

A special case of heterogeneous graphs is given by *multipartite graphs*, where edges are only allowed between nodes of different types:

$$(u, \tau_i, v) \in \mathcal{E} \Rightarrow u \in \mathcal{V}_j, v \in \mathcal{V}_k, j \neq k. \quad (1.135)$$

In multiplex graphs, the graph is decomposed into a collection of  $k$  layers. Each node is assumed to appear in every layer, and each layer corresponds to a distinct relation type, representing intra-layer connectivity [23].

Formally, each layer defines an adjacency matrix corresponding to a specific relation. In addition to intra-layer edges, inter-layer edges may also exist, typically connecting the same node across different layers [23].

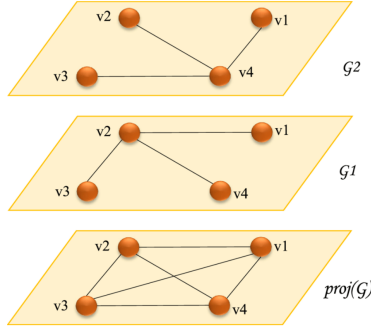


Figure 1.4: A Multiplex Graph with 2 edge types [25].

There are mainly two different levels of GNN tasks. One is the node-level task (e.g., node-level regression or classification). The other is the graph-level task (e.g., graph-level regression or classification). The main difference between them is that one focuses on node-label prediction and the other on graph-label prediction.

The basic GNN message passing is defined as

$$\mathbf{h}_u^{(k)} = \sigma \left( \mathbf{W}_{\text{self}}^{(k)} \mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)} \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)} \right), \quad (1.136)$$

where  $u$  indicates the node index, and  $v$  indicates the neighbor node.  $\mathcal{N}(u)$  indicates the set of neighbors of node  $u$  at this layer.  $\mathbf{W}_{\text{self}}^{(k)}, \mathbf{W}_{\text{neigh}}^{(k)} \in \mathbb{R}^{d^{(k)} \times d^{(k-1)}}$  are trainable parameter matrices at layer  $k$  and  $\sigma$  denotes an elementwise non-linear activation function

(e.g., tanh or ReLU).  $\mathbf{h}_u$  indicates the representation of node  $u$  at layer  $k$ , and  $\mathbf{h}_v$  indicates the representation of node  $v$  at layer  $k$ .  $\mathbf{b}^{(k)} \in \mathbb{R}^{d^{(k)}}$  indicates the bias term. Throughout, superscripts are used to differentiate parameters, embeddings, and dimensionalities across layers [23].

Note that  $\mathbf{W}_{\text{self}}^{(k)}$  transforms the node's own representation from the previous layer  $\mathbf{h}_u^{(k-1)}$ . The term  $\mathbf{W}_{\text{self}}^{(k)} \mathbf{h}_u^{(k-1)}$  represents the contribution of the node's own features when computing its updated representation.  $\mathbf{W}_{\text{neigh}}^{(k)}$  transforms the aggregated representations of the neighboring nodes  $\sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(k-1)}$ . The term  $\mathbf{W}_{\text{neigh}}^{(k)} \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(k-1)}$  represents the contribution of the neighboring nodes' information in updating the representation of node  $u$ .

Equivalently, we can define the basic GNN through the AGGREGATE and UPDATE functions:

$$\text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)} \mid v \in \mathcal{N}(u)\}) = \mathbf{m}_{\mathcal{N}(u)}^{(k)} = \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(k)}, \quad (1.137)$$

$$\text{UPDATE}^{(k)}(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)}) = \sigma \left( \mathbf{W}_{\text{self}}^{(k)} \mathbf{h}_u^{(k)} + \mathbf{W}_{\text{neigh}}^{(k)} \mathbf{m}_{\mathcal{N}(u)}^{(k)} + \mathbf{b}^{(k)} \right), \quad (1.138)$$

where the aggregate function  $\text{AGGREGATE}^{(k)}$  can be customized in many ways, such as sum, mean, maximum, minimum, etc.

Message passing can be at either the node or the graph level, depending on the purpose of the GNN task. For a basic GNN at the graph-level, we can write

$$\mathbf{H}^{(k)} = \sigma \left( \mathbf{A} \mathbf{H}^{(k-1)} \mathbf{W}_{\text{neigh}}^{(k)} + \mathbf{H}^{(k-1)} \mathbf{W}_{\text{self}}^{(k)} \right), \quad (1.139)$$

where  $\mathbf{A}$  is the adjacency matrix. The bias term is omitted for simplicity. This formulation provides a compact representation and highlights efficient implementations via sparse matrix operations [23]. Note that this is the same operation as node-level formulas as described in Equation 1.136.

Note that this equation,

$$\mathbf{H}^{(k)} \in \mathbb{R}^{N \times d_k}, \quad (1.140)$$

denotes the node feature matrix at layer  $k$ , where  $N$  indicates the number of nodes, and  $d_k$  indicates feature dimension at layer  $k$ .

Each row corresponds to a node embedding

$$H^{(k)} = \begin{bmatrix} h_1^{(k)\top} \\ h_2^{(k)\top} \\ \vdots \\ h_N^{(k)\top} \end{bmatrix}, \quad h_i^{(k)} \in \mathbb{R}^{d_k}. \quad (1.141)$$

And self-loops can be defined as

$$\mathbf{h}_u^{(k)} = \text{AGGREGATE} \left( \{ \mathbf{h}_v^{(k-1)} \mid v \in \mathcal{N}(u) \cup \{u\} \} \right), \quad (1.142)$$

where aggregation is now performed over both the neighbors and the node itself. This removes the need for an explicit update function, as the update is implicitly defined through aggregation [23].

Graph neural networks support learning tasks at multiple levels of biological organization, including both node-level and graph-level prediction problems [50]. In node-level tasks, the objective is to infer properties of individual nodes by integrating their features with information from neighboring nodes through message passing. Such tasks are relevant for characterizing single-cell states or identifying functional elements in biological networks. In contrast, graph-level tasks aim to predict global properties of entire networks by aggregating node representations into a unified graph embedding through a readout operation [50]. This enables the analysis of population-level or system-wide behaviors. The ability of GNNs to operate across these scales makes them particularly suitable for studying complex biological systems in which individual-level dynamics give rise to emergent collective phenomena.

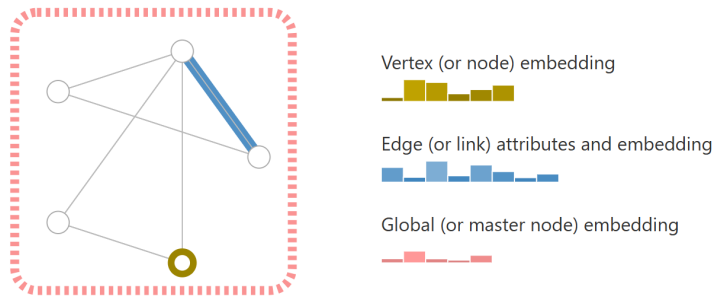


Figure 1.5: Information in the form of scalars or embeddings can be stored at each graph node (left) or edge (right) [50].

### 1.3.4 Activation Function in Graph Neural Network

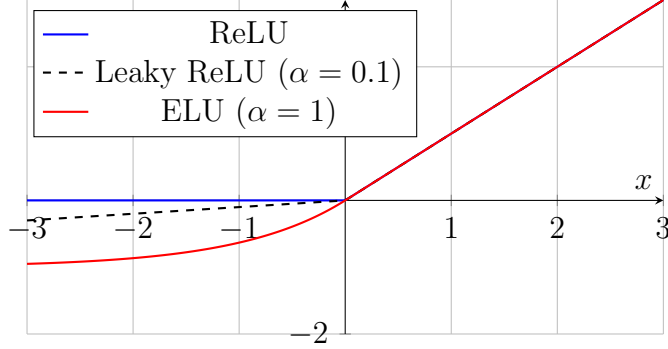


Figure 1.6: Comparison of ReLU, Leaky ReLU, and ELU activation functions. ReLU maps all negative inputs to zero, Leaky ReLU preserves a small linear slope in the negative region, and ELU produces smooth negative outputs. Both Leaky ReLU and ELU help mitigate the vanishing-gradient behavior associated with inactive ReLU units.

Nonlinear activation functions are essential components of deep neural networks, as they allow models to represent complex nonlinear relationships between inputs and outputs. In this work, several commonly used activation functions are considered, including the rectified linear unit (ReLU), the leaky rectified linear unit (Leaky ReLU), and the exponential linear unit (ELU). These activation functions differ mainly in how they treat negative input values.

**ReLU.** The rectified linear unit (ReLU) is one of the most widely used activation functions in deep learning due to its simplicity and computational efficiency [39].

It is defined as

$$\text{ReLU}(x) = \max(0, x). \quad (1.143)$$

Or equivalently,

$$\text{ReLU}(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}. \quad (1.144)$$

For positive inputs, the function behaves linearly, allowing gradients to propagate efficiently. However, for negative inputs, the output is exactly zero. As a result, neurons receiving predominantly negative inputs may stop updating during training, a phenomenon commonly referred to as the *dying ReLU* problem.

**Leaky ReLU.** The leaky rectified linear unit modifies ReLU by allowing a small nonzero slope in the negative region [38].

It is defined as

$$\text{LeakyReLU}(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases}, \quad (1.145)$$

where  $\alpha$  is a small positive constant (commonly  $\alpha = 0.01$  or  $\alpha = 0.1$ ).

Unlike ReLU, negative inputs are not completely suppressed, and therefore gradients can still propagate when  $x \leq 0$ . This modification reduces the likelihood of neurons becoming inactive during training.

**ELU.** The exponential linear unit (ELU) further modifies the behavior of the activation function in the negative region by introducing a smooth nonlinear curve [12].

It is defined as

$$\text{ELU}(x) = \begin{cases} x, & x > 0 \\ \alpha(e^x - 1), & x \leq 0 \end{cases}, \quad (1.146)$$

where  $\alpha > 0$  controls the saturation value for negative inputs. For positive values, ELU behaves identically to ReLU, while negative inputs produce smooth negative outputs that asymptotically approach  $-\alpha$ .

The derivative of ELU is given by

$$\frac{d}{dx}\text{ELU}(x) = \begin{cases} 1, & x > 0 \\ \alpha e^x, & x \leq 0 \end{cases}. \quad (1.147)$$

Because the gradient remains nonzero for negative inputs, ELU helps maintain gradient flow during training. Additionally, the presence of negative outputs helps center the activations around zero, which can improve optimization stability.

ReLU truncates negative inputs to zero, which may lead to inactive neurons. Leaky ReLU alleviates this issue by introducing a small linear slope in the negative region. ELU further improves this behavior by using a smooth exponential curve that preserves gradient information and produces activations with a mean closer to zero. These properties often make ELU advantageous in deep architectures such as graph neural networks. Thus, this project applies ELU as the nonlinear activation function inside GNNs.

### 1.3.5 Graph Convolutional Network and Graph Attention Network

The most basic neighborhood aggregation operation simply computes the sum of neighbor embeddings. One issue with this approach is that it can be unstable and highly sensitive to node degrees [23]. For instance, suppose node  $u$  has significantly more neighbors than node  $u'$ . Then we expect

$$\left\| \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v \right\| \gg \left\| \sum_{v \in \mathcal{N}(u')} \mathbf{h}_v \right\|, \quad (1.148)$$

for any reasonable vector norm  $\|\cdot\|$ . This discrepancy in magnitude can lead to numerical instabilities and difficulties during optimization.

A simple remedy is to normalize the aggregation operation based on node degrees [23]. The most straightforward approach is to take the average rather than the sum

$$\mathbf{m}_{\mathcal{N}(u)} = \frac{1}{|\mathcal{N}(u)|} \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v. \quad (1.149)$$

But alternative normalization strategies have also proven effective. In particular, the symmetric normalization introduced by Kipf and Welling [29] is widely used.

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \frac{\mathbf{h}_v}{\sqrt{|\mathcal{N}(u)| |\mathcal{N}(v)|}}. \quad (1.150)$$

One of the most popular baseline graph neural network models, the Graph Convolutional Network (GCN), employs symmetric normalization together with the self-loop formulation [23]. As described in Equation 1.136, the message passing rule is given by

$$\mathbf{h}_u^{(k)} = \sigma \left( \mathbf{W}_{\text{self}}^{(k)} \mathbf{h}_u^{(k-1)} + \mathbf{W}^{(k)} \sum_{v \in \mathcal{N}(u) \cup \{u\}} \frac{\mathbf{h}_v}{\sqrt{|\mathcal{N}(u)| |\mathcal{N}(v)|}} + \mathbf{b}^{(k)} \right). \quad (1.151)$$

In addition to general aggregation schemes, a powerful strategy for improving GNN performance is to incorporate *attention* mechanisms [3]. The key idea is to assign a learnable importance weight to each neighbor, allowing the model to weigh contributions adaptively [23].

The Graph Attention Network (GAT) [57] applies this idea by defining a weighted aggregation

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \alpha_{u,v} \mathbf{h}_v, \quad (1.152)$$

where  $\alpha_{u,v}$  denotes the attention coefficient assigned to neighbor  $v$  when updating node  $u$  [23].

Then the GAT message-passing can be expressed as follows. For layer  $k$ , we define

$$z_i^{(k)} = W^{(k)} h_i^{(k)} + b^{(k)}, \quad (1.153)$$

where  $h_i^{(k)}$  indicates the embedding of node  $i$  as input vector in layer  $k$ ;  $z_i^{(k)}$  indicates the embedding of node  $i$  in layer  $k$ ;  $W^{(k)}$  and  $b^{(k)}$  are the weights and biases in layer  $k$ .

To determine the attention score of edge  $(i, j)$  in layer  $k$ ,

$$e_{ij}^{(k)} = \text{LeakyReLU} \left( a^{(k)\top} \left[ z_i^{(k)} \parallel z_j^{(k)} \right] \right), \quad (1.154)$$

where  $a^{(k)}$  indicates the learnable weight vector used to compute attention scores in the next equation, and it is part of the GAT layer parameters;  $a^{(k)}$  decides how important neighbor  $j$  is to node  $i$  at layer  $k$ ;  $z_j^{(k)}$ ,  $z_i^{(k)}$  indicates the embedding of node  $j$  and  $i$  in layer  $k$ .  $\parallel$  indicates the concatenation of the node embedding  $z_i^{(k)}$  and  $z_j^{(k)}$  to keep both information intact, and let a neural layer learn how to use them.

To determine the attention weight of edge  $(i, j)$  in layer  $k$ , we normalize the exponential of attention scores,

$$\alpha_{ij}^{(k)} = \frac{\exp(e_{ij}^{(k)})}{\sum_{m \in \mathcal{N}(i)} \exp(e_{im}^{(k)})}. \quad (1.155)$$

We aggregate the neighbor information and apply the GAT message-passing equation as Equation 1.152 to get

$$h_i'^{(k+1)} = \sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(k)} z_j^{(k)}. \quad (1.156)$$

Then we apply the non-linearity to determine the node embedding as input in layer  $k + 1$  as

$$h_i^{(k+1)} = \text{ELU}(h_i'^{(k+1)}). \quad (1.157)$$

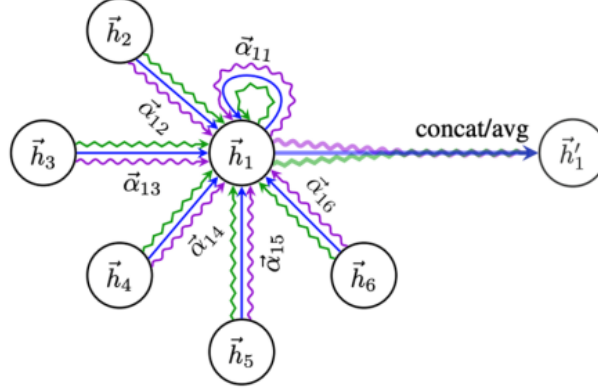


Figure 1.7: Attention on a graph node [30].

### 1.3.6 Hyperparameter Tuning of Graph Attention Network

In this project, GATs are applied more often than other GNN variants. Several hyperparameters in GATs relate to architecture and optimization and significantly impact the overall performance, stability, and generalization of these models. To train these models and make reliable inferences, the selection of hyperparameters must be appropriate. This section summarizes the key hyperparameters that can be tuned in GATs and the approaches to configuring them.

**Model Architecture Parameters.** The expressive capacity of a Graph Attention Network (GAT) model is primarily determined by its architectural hyperparameters [58].

1. **Number of layers.** The depth of a GAT, indicated by  $L$ , controls the size of the receptive field. In practice, most applications employ two to four layers, as deeper architectures may suffer from over-smoothing and optimization difficulties.
2. **Hidden dimension.** The dimensionality of node embeddings in the hidden layers determines the model's representational capacity. Typical values range from 16 to 256, depending on the dataset's size and complexity.
3. **Number of attention heads.** Multi-head attention is a central component of GATs. Let  $K$  denote the number of attention heads. Common choices include  $K \in \{1, 2, 4, 8\}$ . Increasing the number of heads improves expressiveness and stabilizes

training, but also increases computational cost. In the original GAT architecture, eight heads were used in the first layer and a single head in the output layer.

4. **Head aggregation.** Attention heads may be concatenated or averaged. Concatenation increases feature dimensionality and model capacity, whereas averaging preserves dimensionality and reduces the number of parameters.

**Regularization Parameters.** Regularization is critical to prevent overfitting, especially in attention-based models [60].

1. **Feature dropout.** Dropout applied to node features is commonly used, with rates typically ranging from 0.0 to 0.7. That means for each node embedding  $h_j$ , some of its features are randomly selected and reset to zero, and the number of dropout features is also random.
2. **Attention dropout.** Dropout may also be applied to normalized attention coefficients. That means for each attention coefficient  $\alpha_{ij}$ , some of its neighbor attentions are randomly selected and reset to zero, and the number of dropout attentions is also random.
3. **Weight decay.**  $\ell_2$  regularization is often applied to model parameters, with typical values between  $10^{-5}$  and  $10^{-3}$ . It adds a penalty term in the loss function to avoid large weights. Original loss is defined as

$$L(W), \tag{1.158}$$

with weight decay

$$L_{\text{total}} = L(W) + \lambda \|W\|^2, \tag{1.159}$$

where

$$\lambda > 0 \quad (\text{regularization strength}), \tag{1.160}$$

and

$$\|W\|^2 = \sum_{i,j} W_{ij}^2. \tag{1.161}$$

**Optimization Parameters.** Training performance is strongly influenced by the choice of optimization hyperparameters [18].

1. **Learning rate.** GAT models are typically trained using the Adam optimizer. Effective learning rates usually lie in the range  $10^{-4}$  to  $10^{-2}$ , with values between 0.001 and 0.01 being most common.
2. **Early stopping.** To prevent overfitting, early stopping based on validation loss is widely adopted. Patience values between 50 and 200 epochs are commonly used.
3. **Batch size.** For inductive learning or graph-level tasks, batch sizes of 1-32 are typical, depending on graph size and memory constraints.

**Attention Mechanism Parameters.** Several hyperparameters are specific to the attention mechanism [59].

1. **Negative slope of LeakyReLU.** The attention scoring function commonly employs a LeakyReLU activation. The negative slope parameter  $\alpha$  is typically set to 0.1-0.3, with 0.2 as a common default.  $\alpha$  controls how much negative values are preserved.
2. **Self-loops.** Self-loops are usually added to ensure that each node attends to its own features. Most implementations include them by default.
3. **Residual connections.** Residual connections may be introduced between layers to improve gradient flow and training stability, particularly in deeper architectures. That means at layer  $k$  we add  $h_u^{(k)}$  to Equation 1.136, such that we have

$$\mathbf{h}_u^{(k)} = \sigma \left( \mathbf{W}_{\text{self}}^{(k)} \mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)} \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)} \right) + \mathbf{h}_u^{(k)}. \quad (1.162)$$

In summary, the hyperparameters of GATs are critical for a reliable inference from graphs. Therefore, hyperparameter tuning should be performed carefully and systematically to obtain reliable results for graph-based learning tasks. If computational resources are limited, using hyperparameters tuned to the deep learning background will likely improve the overall performance of the GAT.

### 1.3.7 Over-smoothing Issues and Dropout Approaches in Graph Neural Network

**Over-smoothing Problem.** Over-smoothing in graph neural networks arises when repeated message passing causes node representations to become increasingly similar across the graph [37]. As the number of layers increases, the aggregation of neighbor information leads to excessive feature mixing, reducing the discriminative power of node embeddings. Dropout-based regularization methods introduce stochasticity into the message-passing process and can help mitigate this phenomenon.

**Feature Dropout.** As mentioned in subsection 1.3.6, a common regularization technique is feature dropout, in which elements of node feature vectors are randomly masked during training [55].

Let  $H^{(l)} \in \mathbb{R}^{N \times d_l}$  denote the node feature matrix at layer  $l$ .

Feature dropout applies a random binary mask  $M$  such that

$$\tilde{H}^{(l)} = M \odot H^{(l)}, \quad (1.163)$$

where  $\odot$  denotes element-wise multiplication and each entry of  $M$  is sampled as

$$M_{ij} \sim \text{Bernoulli}(1 - p), \quad (1.164)$$

with dropout probability  $p$ .

The message passing update becomes

$$H^{(l+1)} = \sigma\left(\tilde{A}\tilde{H}^{(l)}W^{(l)}\right), \quad (1.165)$$

where  $\tilde{A}$  is the normalized adjacency matrix and  $W^{(l)}$  is the learnable weight matrix. This looks like a compact version of Equation 1.139 because  $W_{neigh}$  and  $W_{self}$  are combined as  $W$  here, and  $\tilde{A} = A + I$ .

Feature dropout helps prevent over-reliance on specific input features but does not directly modify the graph structure.

**Edge Dropout.** A more direct approach to mitigating over-smoothing is edge dropout, commonly referred to as DropEdge [45].

Instead of masking node features, a subset of graph edges is randomly removed during each training iteration.

Let  $A$  denote the adjacency matrix and  $E$  the set of edges. A random subset  $E' \subset E$  is sampled, producing a sparsified adjacency matrix  $\tilde{A}'$ .

The message passing update then becomes

$$H^{(l+1)} = \sigma\left(\tilde{A}' H^{(l)} W^{(l)}\right). \quad (1.166)$$

By reducing the number of edges used during propagation, DropEdge limits the amount of information aggregated at each layer, thereby slowing down the homogenization of node representations and enabling deeper GNN architectures.

**Node Dropout.** Node dropout removes entire nodes (and their incident edges) from the graph during training [24].

Let  $V$  denote the set of nodes in the graph. A random subset  $V' \subset V$  is sampled to construct a subgraph

$$G' = (V', E'), \quad (1.167)$$

where  $E'$  contains edges whose endpoints both lie in  $V'$ . Message passing is then performed on this subgraph.

This technique introduces structural randomness and prevents the model from relying excessively on particular nodes.

**Message Dropout.** Another strategy is message dropout, where the messages transmitted between nodes are randomly suppressed [10].

Suppose the message from node  $j$  to node  $i$  is

$$m_{ij} = W h_j^{(l)}. \quad (1.168)$$

A random mask is applied to the message

$$\tilde{m}_{ij} = z_{ij} m_{ij}, \quad z_{ij} \sim \text{Bernoulli}(1 - p). \quad (1.169)$$

The node update rule becomes

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \tilde{m}_{ij}\right). \quad (1.170)$$

By randomly suppressing messages, this approach reduces the likelihood that nodes repeatedly aggregate identical neighborhood information.

**Effect on Over-Smoothing.** The over-smoothing phenomenon can be interpreted through repeated multiplication by a normalized adjacency matrix [37].

Ignoring nonlinearities and learnable parameters, a simplified GNN update can be written as

$$H^{(L)} \approx \tilde{A}^L X, \quad (1.171)$$

where  $X$  denotes the initial node features.

As  $L$  increases, the operator  $\tilde{A}^L$  progressively mixes node features across the graph. In that case, the features become increasingly averaged, and nodes become more similar to each other. Eventually, all node embeddings become identical.

Dropout-based methods disrupt this process by introducing stochastic perturbations into the propagation operator.

For example, when DropEdge is applied, each layer uses a different adjacency matrix  $\tilde{A}_l$

$$H^{(l+1)} = \sigma\left(\tilde{A}_l H^{(l)} W^{(l)}\right), \quad (1.172)$$

where  $\tilde{A}_l$  is randomly sampled at layer  $l$ . This stochastic propagation prevents repeated application of the same smoothing operator and helps preserve diversity among node embeddings. That means instead of using the same matrix  $\tilde{A}^L$  every time, each layer uses a different random graph.

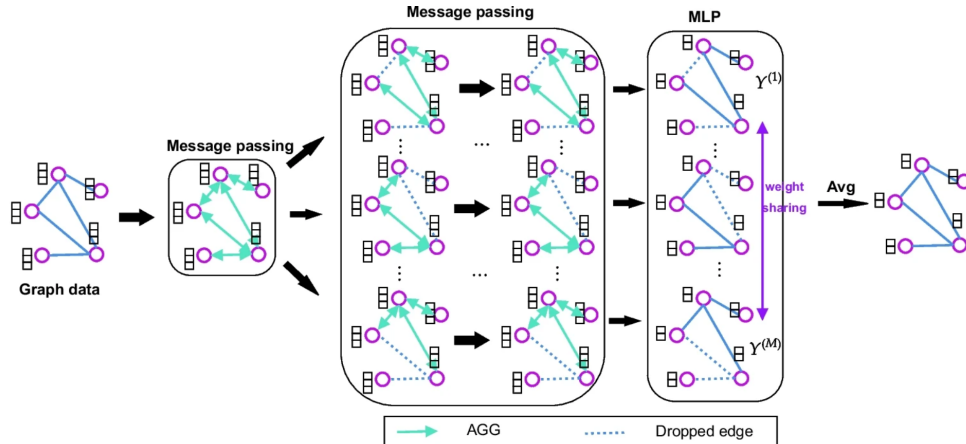


Figure 1.8: Illustration of a DropEdge for network learning [27].

### 1.3.8 Batch Size

In stochastic gradient-based optimization, model parameters are updated using an estimate of the gradient of the loss function. Let  $\theta$  denote the model parameters and  $L(\theta)$  the empirical loss over the entire dataset. The ideal gradient update is

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta} L(\theta_t), \quad (1.173)$$

where  $\eta$  is the learning rate [9].

Computing the full gradient over the entire dataset can be computationally expensive. Therefore, mini-batch stochastic gradient descent (SGD) is commonly used. Given a batch  $B$  containing  $|B|$  samples, the gradient is approximated as

$$g_B = \frac{1}{|B|} \sum_{i \in B} \nabla_{\theta} \ell_i(\theta), \quad (1.174)$$

where  $\ell_i(\theta)$  denotes the loss for the  $i$ -th sample [21].

This estimator is unbiased

$$\mathbb{E}[g_B] = \nabla_{\theta} L(\theta). \quad (1.175)$$

However, the variance of the gradient estimate depends on the batch size

$$\text{Var}(g_B) \propto \frac{\sigma^2}{|B|}, \quad (1.176)$$

where  $\sigma^2$  represents the variance of the individual sample gradients [9, 21].

Consequently, smaller batch sizes yield noisier gradient estimates, leading to more stochastic optimization trajectories [54]. This often leads to oscillatory training curves, but can improve generalization by helping the optimizer escape sharp minima. In contrast, larger batch sizes reduce the variance of the gradient estimate, yielding smoother and more stable training dynamics.

In practice, the choice of batch size introduces a trade-off between computational efficiency, optimization stability, and generalization performance. Smaller batch sizes introduce greater stochasticity in parameter updates, whereas larger batch sizes yield more accurate gradient estimates, but may reduce the regularization effect of gradient noise.

Note that only graph-level data can be split into batches for training, whereas node-level data cannot, because batching destroys inter-node connectivity and the graph structure.

## 1.4 Experimental Setup

A complete pipeline for experiments was previously developed by Atiyeh Ahmadi et al. to calibrate ABMs of microbial populations, covering the entire process from the collection of real experimental images to model parameterization [2]. In this project, the same procedure is followed with some modifications, and the experimental data are provided by Atiyeh Ahmadi et al. [2]. Two essential packages are applied. **CellProfiler**, a tool for exploring and analyzing high-dimensional image data, and **TrackRefiner**, a tool designed for the tracking and refinement of bacterial data, are applied to extract 69 features representing various biophysical properties of microcolonies [56, 1].

Cell morphology and orientation were quantified in **CellProfiler** by fitting ellipses to individual cell outlines. Cell length was defined as the major axis of the fitted ellipse, while orientation was measured as the angle between this axis and the horizontal reference direction. The centroid of each cell was taken as the center of the fitted ellipse. The elongation rates were estimated from the slope of a linear regression of the logarithm of cell length on time. The cell radius was calculated as the average length of the minor axes from birth to division. The length of the division was defined as the cell length measured in the frame immediately preceding the division [2].

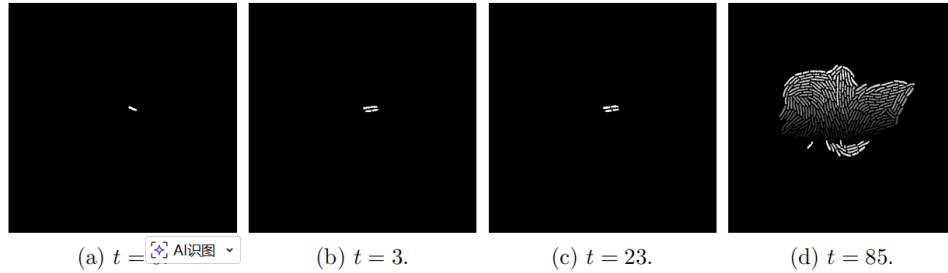


Figure 1.9: Experimental samples of Time-lapse images from the first frame to the last frame [13]. This figure is from Sushma Dhamodharan, one of the previous master’s students in our research group. The experimental images were taken by my senior, Atiyeh Ahmadi [2].

## 1.5 Simulation Module Setup

Agent-based simulations were performed using the **CellModeller** framework [47], in which individual cells are represented as rigid, capsule-shaped objects that undergo exponential

elongation at a rate proportional to their current length. Cellular motion and spatial rearrangement were determined using an energy-minimization procedure that enforces quasi-static mechanical equilibrium. The simulation module setup is proposed by Atiyeh Ahmadi et al. as follows [2]. Note that **CellModeller** is an open-source **Python** package, making it easy to use and apply.

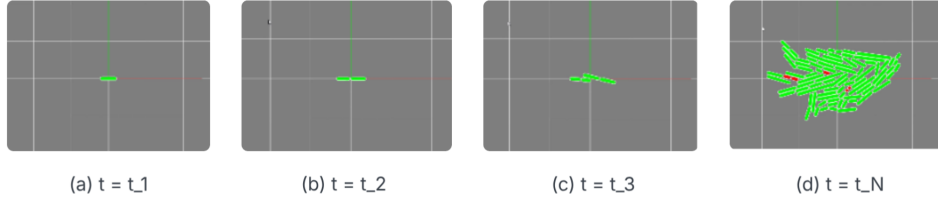


Figure 1.10: Simulated sample images from timestep  $t = t_1$  to timestep  $t = t_N$ . This visualization mimics time-lapse images from real experiments. The timestep  $t_1 < t_2 < t_3 < t_N$ , but it's not assumed that  $t_{(i-1)} + 1 = t_i$  for all the  $i$  from 1 to  $N$ . This means that these four images are arbitrarily selected from the simulation videos and are not necessarily sequential in time.

In the **CellModeller** simulation module, there are two types of parameters that play key roles: observable and unobservable parameters. Observable parameters are those that can be observed in real experiments through specific measurements. Unobservable parameters are those that cannot be inferred from experiments directly and must be inferred by running the calibration pipeline. In a calibration task, observable parameters are typically controlled and fixed at specified values or distributions, while unobservable parameters are calibrated from experiments to infer underlying information about them.

Two key parameters governing the model's mechanical behavior were selected for detailed analysis [2]. The parameter  $\alpha$  controls the penalty for cell displacement during overlap resolution, thus regulating the effective rigidity of the cells. The lower values of  $\alpha$  correspond to more rigid behavior, while the higher values allow for increased deformation and unresolved overlap. The parameter  $\gamma$  represents the inhibition of growth resulting from neighboring cells and other physical constraints. Higher values of  $\gamma$  enable cells to overcome local restrictions and continue to elongate, while lower values impose stronger growth limitations. Although the default implementation in **CellModeller** enforces the relation  $\alpha = 1/\gamma$ , reflecting an inverse dependence between overlap tolerance and rigidity, these parameters were calibrated independently in the present study to enable a broader range of mechanical behaviors.

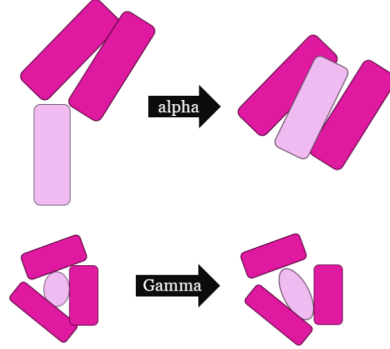


Figure 1.11: ABM parameters  $\alpha$  and  $\gamma$ , the left side indicates when  $\alpha$  and  $\gamma$  are low, the right side shows how the displacement and growth affect the cells and their neighbors as the value increases [13]. This figure is from Sushma Dhamodharan, one of the previous master's students in our research group.

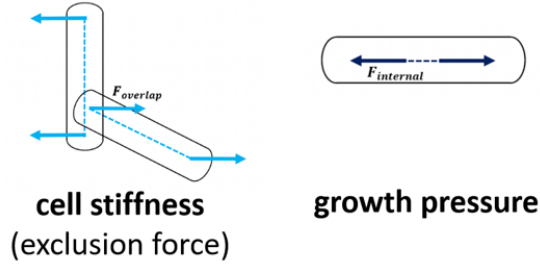


Figure 1.12: ABM parameters  $\alpha$  (left) and  $\gamma$  (right). This figure was done by my seniors.

The parameters describing the cell radius ( $r$ ), maximum elongation rate ( $\mu_{\max}$ ), and division length threshold ( $L_{\text{div}}$ ) were estimated by fitting probability distributions to experimental measurements obtained from 24 experimental samples [2]. Growth rates that are observed below  $0 \text{ h}^{-1}$  and above  $2.1 \text{ h}^{-1}$ , corresponding to non-physiological shrinking and unrealistically rapid doubling, respectively, were excluded as outliers. The division length thresholds assigned to the daughter cells were sampled from a log-normal distribution fitted to empirical data, with a lower cutoff of  $2 \mu\text{m}$  to prevent unrealistically small cell sizes. Maximum growth rates were drawn similarly from a normal distribution that matched the experimental observations. All simulated cells were assigned a uniform radius equal to the experimental mean, consistent with the low variability observed in measured radii (standard deviation = 0.07). The 0.025 hour is defined as one time step, which is the

simulation time increment used to update the system state, and the parameter *pickSteps* (or pickle step) indicates the number of time steps between every 2 simulation state records.

A summary of the model parameters and variables is provided in the following table [2].

Table 1.1: `CellModeller` settings.

| Name                         | Value | Units           |
|------------------------------|-------|-----------------|
| $r$ mean                     | 0.67  | $\mu\text{m}$   |
| $L_{div}$ scale parameter    | 4.80  | $\mu\text{m}$   |
| $L_{div}$ shape parameter    | 0.42  | $\mu\text{m}$   |
| $L_{div}$ location parameter | -0.32 | $\mu\text{m}$   |
| $\mu_{\max}$ mean            | 1.15  | $\text{h}^{-1}$ |
| $\mu_{\max}$ s.d.            | 0.31  | $\text{h}^{-1}$ |
| Time step                    | 0.025 | $\text{h}$      |

## 1.6 Conventional Approximate Bayesian Computation

### 1.6.1 Bayesian Inference

Bayesian inference is a statistical framework for estimating unknown parameters by combining prior knowledge with observed data using Bayes' theorem. Bayesian inference treats unknown parameters as random variables and provides a full probability distribution over their possible values. It updates prior beliefs using data to obtain a posterior distribution that quantifies uncertainty in model parameters and predictions [19].

Formally, for parameters  $\theta$  and observed data  $y$ , Bayes' theorem says that the posterior is given by

$$\pi(\theta | y) = \frac{\pi(\theta) p(y | \theta)}{p(y)}, \quad (1.177)$$

where  $\pi(\theta)$  is the prior,  $p(y)$  is the evidence, and  $p(y | \theta)$  is the likelihood.

We only care about relative values, and evidence is hard to compute, so we focus on

$$\pi(\theta | y) \propto \pi(\theta) p(y | \theta). \quad (1.178)$$

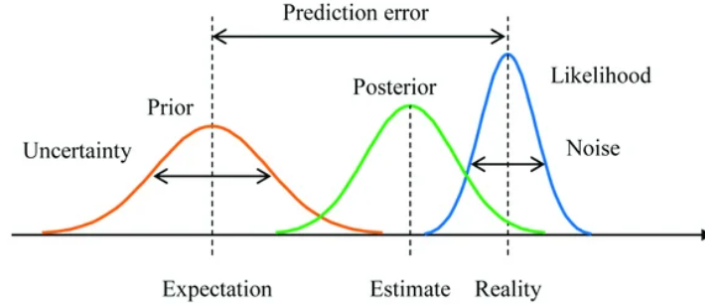


Figure 1.13: Example of Bayesian inference with a prior distribution, a posterior distribution, and a likelihood function. The prediction error is the difference between the prior expectation and the peak of the likelihood function (i.e., reality). Uncertainty is the variance of the prior. Noise is the variance of the likelihood function. [61].

### 1.6.2 Approximate Bayesian Computation Sequential Monte Carlo

Computational Bayesian inference addresses this intractable likelihood issue by employing numerical algorithms, including Markov chain Monte Carlo (MCMC) and likelihood-free methods, to approximate posterior distributions [44].

Approximate Bayesian Computation (ABC) is one of the likelihood-free methods. It addresses this issue by enabling simulation-based comparisons (between model outputs and experimental data) rather than requiring an explicit likelihood evaluation [4]. It computationally approximates the true posterior  $p(y | \theta)$  by running simulations.

Sequential Monte Carlo (SMC) approximates posterior distributions by iteratively updating weighted populations of samples [15]. Approximate Bayesian Computation sequential Monte Carlo (ABC-SMC) is a variant of ABC that improves efficiency [53].

In ABC, the intractable likelihood  $p(y | \theta)$  is replaced by a simulation-based comparison [5, 52].

Given observed data  $y$  and simulated data  $x \sim p(x | \theta)$ , similarity is quantified through a distance function

$$d(x, y). \quad (1.179)$$

This  $d$  can be the Euclidean distance as

$$d(x, y) = \|x - y\|_2, \quad (1.180)$$

which is simple but sensitive to scaling and ignores correlations [5].

Or  $d$  can be the Mahalanobis distance as

$$d(x, y) = \sqrt{(x - y)^T \Sigma^{-1} (x - y)}, \quad (1.181)$$

which accounts for correlations between components and is often preferred when summary statistics are approximately Gaussian [52].

In many applications, especially when the data  $y$  and  $x$  are high-dimensional, comparing raw data directly is computationally inefficient or statistically unstable.

We specify functions  $S$  that can extract useful information from simulated data  $x$  and experimental data  $y$ , so that the dimensionality is reduced, but the information about the parameters is kept. Such functions  $S$  are called summary statistics because they summarize information into statistics. Then ABC approximates the true posterior  $p(\theta \mid y)$  by  $p(\theta \mid S(y))$ .

Other dimensionality reduction methods, such as PCA, can always lose crucial information about parameters. Ideally, we want to reduce the data dimension by removing noise and saving information with respect to parameters

ABC relies on summary statistics,

$$d(x, y) \longrightarrow d(S(x), S(y)), \quad (1.182)$$

where the summary statistics  $S$  map the data to a lower-dimensional feature space.

To efficiently approximate the posterior, ABC-SMC applies a sequence of decreasing tolerances,

$$\varepsilon_1 > \varepsilon_2 > \cdots > \varepsilon_T, \quad (1.183)$$

which defines a sequence of intermediate distributions

$$\pi_{\varepsilon_1} \rightarrow \pi_{\varepsilon_2} \rightarrow \cdots \rightarrow \pi_{\varepsilon_T}. \quad (1.184)$$

A generation is one round of  $N$  particles that approximates the posterior under a specific tolerance  $\varepsilon_t$ .

At generation  $t$ , we represent the distribution using weighted particles

$$\left\{ \theta_t^{(i)}, w_t^{(i)} \right\}_{i=1}^N, \quad (1.185)$$

where  $t = 1, \dots, T$  indicates the generation index, and  $i = 1, \dots, N$  indicates the particle index.

To initialize the first generation  $t = 1$ , for each particle  $i = 1, \dots, N$ :

1. Sample

$$\theta_1^{(i)} \sim \pi(\theta). \quad (1.186)$$

2. Simulate

$$x \sim p(x \mid \theta_1^{(i)}), \quad (1.187)$$

where  $y$  are observed data.

3. Accept if

$$d(x, y), \text{ or } d(S(x), S(y)) \leq \varepsilon_1, \quad (1.188)$$

where  $x$  are simulated data.

4. Assign weights

$$w_1^{(i)} = \frac{1}{N}. \quad (1.189)$$

We define the proposal distribution as a mixture

$$q_t(\theta) = \sum_{j=1}^N w_{t-1}^{(j)} K_t(\theta \mid \theta_{t-1}^{(j)}), \quad (1.190)$$

where  $K_t$  is a perturbation kernel. This takes all the values of the accepted parameters, perturbs them, and assigns them different weights. In this case, multiple hypotheses can be maintained simultaneously.

If there is more than 1 generation, it is called sequential generations. For each particle  $i = 1, \dots, N$ :

1. Sample an index

$$j \sim \text{Categorical}(w_{t-1}^{(1)}, \dots, w_{t-1}^{(N)}), \quad (1.191)$$

where  $j$  indicates an index of a particle from the previous generation  $t - 1$ .

Recall that a categorical distribution is the generalization of a Bernoulli distribution to multiple discrete outcomes. So, it means choosing one index  $j$  from  $N$  possible categories with probabilities given by weights.

2. Perturb

$$\theta^* \sim K_t(\cdot \mid \theta_{t-1}^{(j)}). \quad (1.192)$$

3. Simulate

$$x \sim p(x \mid \theta^*). \quad (1.193)$$

4. Accept if

$$d(x, y), \text{ or } d(S(x), S(y)) \leq \varepsilon_t. \quad (1.194)$$

5. Set

$$\theta_t^{(i)} = \theta^*. \quad (1.195)$$

The importance weights are given by

$$\tilde{w}_t^{(i)} = \frac{\pi(\theta_t^{(i)})}{\sum_{j=1}^N w_{t-1}^{(j)} K_t(\theta_t^{(i)} \mid \theta_{t-1}^{(j)})}. \quad (1.196)$$

They are normalized as

$$w_t^{(i)} = \frac{\tilde{w}_t^{(i)}}{\sum_{k=1}^N \tilde{w}_t^{(k)}}, \quad (1.197)$$

where  $\tilde{w}_t^{(i)}$  denotes the unnormalized weight,  $w_t^{(i)}$  denotes the normalized weight.

### 1.6.3 Importance Weights

To understand the importance weights, it is important to introduce importance sampling, which is applied to reduce the sampling bias [53]. Sampling bias occurs when samples are drawn from the wrong proposed distribution, which is far from the true distribution. The importance sampling says

$$\text{weight} = \frac{\text{target distribution}}{\text{proposal distribution}}. \quad (1.198)$$

At step  $t$ , the target distribution is

$$\pi_{\epsilon_t}(\theta \mid y) \propto \pi(\theta) \mathbb{P}(d_\theta(x, y) \leq \epsilon_t \mid \theta). \quad (1.199)$$

Here, it is clear that ABC computationally approximates the likelihood  $p(y \mid \theta)$  by  $\mathbb{P}(d_\theta(x, y) \leq \epsilon_t \mid \theta)$  (i.e., running simulations and taking the ones close enough to observations). We only care about the proportionality here, so the evidence is ignored.

Equivalently, this can be written as

$$\pi_{\epsilon_t}(\theta \mid y) \propto \pi(\theta) \int \mathbf{1}[d_\theta(x, y) \leq \epsilon_t] p(x \mid \theta) dx, \quad (1.200)$$

where this integral cannot be computed, it can only be estimated by simulations. Or

$$\mathbb{P}(d_\theta(x, y) \leq \epsilon_t \mid \theta) = \mathbb{P}(d_\theta(x, y) \leq \epsilon_t \mid x \sim p(x \mid \theta)). \quad (1.201)$$

However, in ABC we do not explicitly evaluate the probability

$$\mathbb{P}(d_\theta(x, y) \leq \epsilon_t \mid \theta). \quad (1.202)$$

Instead, this condition is enforced implicitly by accepting only those simulated samples that satisfy the tolerance criterion. That being said,  $\mathbb{P}(d_\theta(x, y) \leq \epsilon_t \mid \theta)$  is not evaluated explicitly. The condition is enforced by accepting only samples with  $d_\theta(x, y) \leq \epsilon_t$ , such that it enforces

$$\mathbb{P}(d_\theta(x, y) \leq \epsilon_t \mid \theta) = 1. \quad (1.203)$$

So, for the accepted particles, the effective target distribution simplifies to

$$\pi_{\epsilon_t}(\theta \mid y) \propto \pi(\theta). \quad (1.204)$$

It is clear that

$$w(\theta) = \frac{\pi(\theta)}{q_t(\theta)}. \quad (1.205)$$

Substituting the proposal distribution  $q_t(\theta)$  gives

$$w(\theta) = \frac{\pi(\theta)}{\sum_{j=1}^N w_{t-1}^{(j)} K_t(\theta \mid \theta_{t-1}^{(j)})}. \quad (1.206)$$

Another helpful perspective to understand the importance sampling is think about the goal of it, or the proposal distribution which is defined above. We want something like

$$\mathbb{E}_\pi[f(\theta)] = \int f(\theta) \pi(\theta) d\theta. \quad (1.207)$$

This represents the average value of  $f(\theta)$  under the target distribution  $\pi(\theta)$ . The  $f(\theta)$  means any arbitrary function of the parameter  $\theta$ , so it can be parameter  $\theta$  itself.

We cannot sample from  $\pi(\theta)$  directly. Instead, we sample from another distribution (i.e., the proposal distribution)

$$\theta \sim q(\theta). \quad (1.208)$$

A simple algebra trick comes in

$$\int f(\theta) \pi(\theta) d\theta = \int f(\theta) \frac{\pi(\theta)}{q(\theta)} q(\theta) d\theta = \mathbb{E}_q \left[ f(\theta) \frac{\pi(\theta)}{q(\theta)} \right], \quad (1.209)$$

because

$$\frac{q(\theta)}{q(\theta)} = 1. \quad (1.210)$$

So,

$$\mathbb{E}_\pi[f(\theta)] = \mathbb{E}_q \left[ f(\theta) \frac{\pi(\theta)}{q(\theta)} \right]. \quad (1.211)$$

Then turn this into a Monte Carlo estimator. If we sample

$$\theta_1, \dots, \theta_N \sim q(\theta), \quad (1.212)$$

then

$$\mathbb{E}_\pi[f(\theta)] \approx \frac{1}{N} \sum_{i=1}^N f(\theta_i) \frac{\pi(\theta_i)}{q(\theta_i)} = \frac{1}{N} \sum_{i=1}^N f(\theta_i) w(\theta_i). \quad (1.213)$$

That is because

$$w(\theta_i) = \frac{\pi(\theta_i)}{q(\theta_i)}, \quad (1.214)$$

and obviously it is the same format as

$$q_t(\theta) = \sum_j w_{t-1}^{(j)} K(\theta \mid \theta_j). \quad (1.215)$$

#### 1.6.4 Perturbation Kernel and Effective Sample Size

In terms of the perturbation kernel, a common choice is a Gaussian kernel [\[53\]](#). That is

$$K_t(\theta \mid \theta') = \mathcal{N}(\theta; \theta', \Sigma_t), \quad (1.216)$$

with adaptive covariance

$$\Sigma_t = 2 \text{Cov}_{w_{t-1}}(\theta_{t-1}), \quad (1.217)$$

where

$$\text{Cov}_w(\theta) = \sum_{i=1}^N w^{(i)} (\theta^{(i)} - \bar{\theta})(\theta^{(i)} - \bar{\theta})^\top, \quad \bar{\theta} = \sum_{i=1}^N w^{(i)} \theta^{(i)}. \quad (1.218)$$

All the perturbation kernel does is to take a good parameter and slightly perturb it. For example, if 5 is accepted in generation  $t-1$ , in generation  $t$ , we choose a new parameter that is very close to 5, since we already know it is a good parameter. That being said,  $\theta' = 5 \Rightarrow \theta \sim \mathcal{N}(5, 1)$ .

The effective sample size is defined as

$$\text{ESS}_t = \frac{1}{\sum_{i=1}^N (w_t^{(i)})^2}. \quad (1.219)$$

ESS reflects the mismatch between  $q_t(\theta)$  and  $\pi_{\epsilon_t}(\theta)$ . Low ESS indicates that the proposal distribution is far from the target distribution. If  $\text{ESS}_t$  falls below a threshold, resampling should be performed, and the weights are reset:

$$w_t^{(i)} = \frac{1}{N}. \quad (1.220)$$

The final particle system,

$$\left\{ \theta_T^{(i)}, w_T^{(i)} \right\}_{i=1}^N, \quad (1.221)$$

provides a weighted approximation of the posterior distribution as

$$\pi(\theta \mid y), \quad (1.222)$$

which is effectively approximated as

$$\pi(\theta \mid x). \quad (1.223)$$

### 1.6.5 Distribution-based Distance with Kullback-Leibler Divergence

Let  $\theta$  denote the model parameters with prior distribution

$$\theta \sim \pi(\theta). \quad (1.224)$$

Given a simulator defined implicitly by

$$x \sim p(x \mid \theta). \quad (1.225)$$

We assume that the likelihood  $p(y \mid \theta)$  is intractable, but simulation from the model is available.

Given  $M$  experimental samples  $\{y^{(m)}\}_{m=1}^M$ , we compute the summary statistics

$$s_{\text{exp}}^{(m)} = S(y^{(m)}). \quad (1.226)$$

Then we take their average

$$\bar{s}_{\text{exp}} = \frac{1}{M} \sum_{m=1}^M s_{\text{exp}}^{(m)}. \quad (1.227)$$

Assume we take  $H$  simulated samples for each parameter  $\theta$ . Then given the simulated samples  $\{x^{(h)}\}_{h=1}^H$ , we compute the summary statistics

$$s_{\text{sim}}^{(h)} = S(x^{(h)}). \quad (1.228)$$

We can define the discrepancy between simulated and experimental summaries using the  $\ell_1$  distance. Mathematically,

$$d_{\theta}(x, y) = \|\bar{s}_{\text{sim}} - \bar{s}_{\text{exp}}\|_1, \quad (1.229)$$

and the  $\ell_1$  distance gives less penalty to outliers than the  $\ell_2$  distance.

We can make it a relative  $\ell_1$  distance. Mathematically,

$$d_{\theta}(x, y) = \left\| \frac{\bar{s}_{\text{sim}} - \bar{s}_{\text{exp}}}{\bar{s}_{\text{exp}}} \right\|_1. \quad (1.230)$$

To capture variability and the shape of the parameter distribution in the experimental observations, we apply the Kullback-Leibler (KL) divergence to compare distributions [34].

Now recall that there are two distributions of summary statistics, namely

1. experimental:

$$\{s_{\text{exp}}^{(m)}\}_{m=1}^M. \quad (1.231)$$

2. simulated:

$$\{s_{\text{sim}}^{(h)}(\theta)\}_{h=1}^H. \quad (1.232)$$

Recall the definition of Kernel Density Estimation (KDE) to get the distribution

$$\hat{p}(\theta) = \frac{1}{N} \sum_{i=1}^N K_{\text{Gaussian}}(\theta - \theta^{(i)}), \quad (1.233)$$

where the kernel function  $K_h$  is usually defined as Gaussian

$$K_{\text{Gaussian}}(x) = \frac{1}{h} K\left(\frac{x}{h}\right), \quad (1.234)$$

where  $K(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ , and  $h$  indicates bandwidth that controls how wide the kernel is.

Define empirical distributions of the summary statistics

$$P_{\text{exp}}(s) = \frac{1}{M} \sum_{m=1}^M K_{\text{Gaussian}}(s - s_{\text{exp}}^{(m)}), \quad (1.235)$$

and

$$P_{\text{sim}}(s \mid \theta) = \frac{1}{H} \sum_{h=1}^H K_{\text{Gaussian}}\left(s - s_{\text{sim}}^{(h)}(\theta)\right). \quad (1.236)$$

Then define the discrepancy

$$d(\theta) = D_{\text{KL}}(P_{\text{exp}} \parallel P_{\text{sim}}(\theta)), \quad (1.237)$$

where the KL divergence is defined as

$$D_{\text{KL}}(P \parallel Q) = \int P(s) \log\left(\frac{P(s)}{Q(s)}\right) ds. \quad (1.238)$$

The KL divergence is estimated using the density estimation, but this can be unstable for small sample sizes or high-dimensional summaries.

Then ABC becomes

$$d(\theta) = D_{\text{KL}}(P_{\text{exp}}, P_{\text{sim}}(\theta)). \quad (1.239)$$

The parameter is accepted if

$$d(\theta) \leq \epsilon. \quad (1.240)$$

In general, the distribution-based distance of neural summary statistics can be written as

$$d(\theta) = D_{\text{KL}}(P_{\text{exp}}(S(y)), P_{\text{sim}}(S(x) \mid \theta)). \quad (1.241)$$

### 1.6.6 Previous Work

Atiyeh Ahmadi et al. conducted a previous study on this inference pipeline and reported results that we can refer to [2]. The two unobservable model parameters (gamma and alpha) were estimated using the ABC-SMC framework, implemented with the `pyabc` software package [32]. The algorithmic settings were selected based on the values reported in Table 1.2. In each population, the tolerance threshold  $\varepsilon$  was defined as the median of the distance values obtained from the preceding set of particles. The initial tolerance level was determined by generating 40 accepted particles by logarithmic sampling from prior distributions [2].

Table 1.2: ABC-SMC settings.

| Setting                         | Value          |
|---------------------------------|----------------|
| Replicates per parameter set    | 10             |
| Prior distribution ( $\alpha$ ) | $U(0.01, 100)$ |
| Prior distribution ( $\gamma$ ) | $U(0.1, 1000)$ |
| Accepted simulations for round  | 40             |

In this study,  $D_{KL}$  was calculated using the numerical procedure proposed by Perez-Cruz [41]. For each parameter configuration, ten independent simulation replicas were generated to construct the empirical estimate of  $Q(x)$ . In addition to distribution-based comparisons, the overall discrepancy between experimental and simulated summary statistics was evaluated using the Euclidean distance, with all summary measures assigned equal weights.

This project used 6 biological summary statistics specified based on biological knowledge: microcolony aspect ratio, microcolony convexity, microcolony density, normalized growth rate, agreement with Exponential Growth, and orientational order parameter. The summary statistics are proposed by Atiyeh Ahmadi et al. [2].

Atiyeh Ahmadi et al. performed a global multivariate sensitivity analysis to assess the extent to which the selected summary statistics contribute to model calibration [2]. Parameter values were sampled logarithmically over predefined ranges. For each parameter configuration, an ensemble of 50 independent simulations was initialized from a single cell and executed until the population reached approximately 200 cells. The resulting outputs, summarized in Fig. 3, demonstrate that the chosen statistics provide meaningful discriminative power for identifying the parameters  $\alpha$  and  $\gamma$ .

Analysis of simulation results revealed that microcolony density, convexity, and orientational order increased monotonically with  $\alpha$ , consistent with increased cellular flexibility, facilitating reorientation and spatial rearrangement, thereby promoting efficient packing [2]. In contrast, the aspect ratio of the colony decreased with increasing  $\alpha$ , indicating stronger cellular alignment and preferential elongation along the dominant growth direction.

Variations in the parameter  $\gamma$  exhibited a comparatively weaker influence on colony geometry [2]. Reduced values of  $\gamma$  were associated with modest increases in aspect ratio, suggesting that growth constraints within the colony interior may promote more isotropic expansion by reducing inhibition in less crowded regions.

Both parameters were positively correlated with overall growth rates [2]. Elevated values of  $\alpha$  corresponded to increased elongation, reflecting the beneficial effects of cellular flexibility on population expansion. Similarly, higher values of  $\gamma$ , indicative of reduced growth restriction, were associated with increased proliferation. Deviations from ideal exponential growth exhibited only limited sensitivity to  $\gamma$  for small colonies, but this dependence became more pronounced as the size of the colony increased, reflecting the growing impact of internal crowding on growth dynamics.

Note that more details about the posterior and sensitivity analysis results of the previous case study can be found in [Appendix B](#).

## 1.7 Semi-automatic and Machine Learning-based Approximate Bayesian Computation

Semi-automatic ABC is a method for constructing near-optimal summary statistics using regression. It is called “semi-automatic” because summary statistics are learned automatically, but the training data must be user-defined.

The previous study showed that, under mild conditions, the optimal summary statistic for parameter  $\theta$  and observed data  $y$  is given by

$$S(y) = \mathbb{E}(\theta \mid y). \quad (1.242)$$

That is, the posterior mean, which is generally unknown, is commonly approximated using regression techniques. People did not use deep learning or neural networks at that

time because machine learning was in its early stages of regression and classification. Semi-automatic ABC provided a guarantee of optimal summary statistics for parameters, which is rare in modern papers but still holds [17].

The semi-automatic ABC procedure proposed by Fearnhead and Prangle [17] is summarized as follows.

1. As described in Equation 1.186, sample parameters  $\theta_i \sim \pi(\theta)$ , where  $\pi(\theta)$  is the prior and  $i = 1, \dots, N$  is the training sample index if we assume there are  $N$  training samples.
2. As described in Equation 1.187, simulate synthetic data  $x_i \sim p(x_i \mid \theta_i)$  for all the  $i = 1, \dots, N$ .
3. Fit a regression model of the form

$$\theta_i = f(x_i) + \varepsilon_i, \quad (1.243)$$

where  $f(\cdot)$  denotes a regression function and  $\varepsilon_i$  is an error term for  $i = 1, \dots, N$ .

4. Construct summary statistics using the trained regression function  $f$

$$S(y_j) = f(y_j), \quad (1.244)$$

for all the observed data  $y_j$  where  $j = 1, \dots, M$  is the testing sample index, if we assume there are  $M$  observed samples for testing.

5. Perform ABC inference using summary statistics  $S(y_j)$  and  $S(x_i)$  for all the simulated data  $x_i$  and observed data  $y_j$  where  $i = 1, \dots, N$  and  $j = 1, \dots, M$ .

In semi-automatic ABC, the regression model learns how the data depend on parameters and extracts the most informative components. Summary statistics are obtained by regressing model parameters on simulated data, yielding approximations to posterior expectations. More recent approaches employ neural networks to learn informative embeddings directly from high-dimensional observations [28]. These methods reduce the information loss associated with manually designed statistics and improve inference accuracy, particularly for complex biological models, albeit at the cost of reduced interpretability and increased computational requirements.

The training loss and model performance can indirectly indicate the informativeness of summary statistics because the loss function is specified in a manner similar to mutual information or other metrics to quantify informativeness.

Unlike conventional ABC, semi-auto ABC generates model-dependent summary statistics  $f$ , whereas conventional ABC provides data-dependent summary statistics  $S$ . This is why a model mismatch affects semi-auto ABC results much more than it does in the conventional approach. Model-dependent summary statistics are learned by ML models, so if we train the ML models on one group of data and predict on a very different group, the mismatch kills everything because the learned ML model  $f$  can be informative only if the two groups are not very different. Data-dependent summary statistics are functions that depend on the data, such as the mean or variance.

In semi-automatic ABC, it is easier to evaluate the informativeness of summary statistics because regression uses the predictions as the learned summary statistics. For more details on informativeness, see [section 1.8](#).

There are more machine learning approaches to do this learning, except for regression. Semi-automatic ABC uses regression as an example to demonstrate that the learning metric works. The MSE can be replaced with other loss functions for specific learning tasks, such as cross-entropy for classification.

If there are sufficient experimental samples, we can train the regression model on simulated data and directly predict on the experimental data. There is no reason to apply ABC unless our experimental samples are insufficient. For example, if we have 1000 experimental samples and set 1000 simulators to form the posterior in ABC, there is no difference between making direct predictions and running ABC. ABC helps approximate the true posterior when the number of observations is insufficient.

One potential issue is that if the data are independent of the parameters, then even sufficiently informative summary statistics remain independent of the parameters because summary statistics extract information rather than create it. More details on summary statistics will be given in the next section [section 1.8](#).

## 1.8 Informativeness of Summary Statistics

In ABC, statistics  $S(y)$  are sufficient for  $\theta$  if

$$\pi(\theta | y) = \pi(\theta | S(y)). \quad (1.245)$$

So, sufficiency means no information loss about  $\theta$  [4].

The parameter  $\theta$  is identifiable if

$$p(y | \theta_1) = p(y | \theta_2) \Rightarrow \theta_1 = \theta_2. \quad (1.246)$$

Or

$$p(S(y) \mid \theta_1) = p(S(y) \mid \theta_2) \Rightarrow \theta_1 = \theta_2. \quad (1.247)$$

Or

$$\mathbb{E}[S(y) \mid \theta_1] = \mathbb{E}[S(y) \mid \theta_2] \Rightarrow \theta_1 = \theta_2. \quad (1.248)$$

Identifiability determines whether the parameters uniquely determine the likelihood distributions.

Sensitivity measures how summary statistics respond to parameter changes [11].

$$\frac{\partial}{\partial \theta} \mathbb{E}[S(y) \mid \theta]. \quad (1.249)$$

Sensitivity measures how strongly summaries respond to changes in parameters [11].

Informativeness is a concept rather than a single fixed quantity [4]. It determines how much information about the parameter  $\theta$  is preserved. There are multiple ways to quantify this.

Mutual Information measures how much uncertainty about  $\theta$  is reduced by  $S(y)$ . Mathematically,

$$I(\theta; S(y)) = H(\theta) - H(\theta \mid S(y)). \quad (1.250)$$

Equivalently,

$$I(\theta; S(y)) = - \int p(\theta) \log p(\theta) d\theta + \mathbb{E}_{\theta, S(y)}[\log p(\theta \mid S(y))]. \quad (1.251)$$

So,

$$I(\theta; S) = \int p(\theta, s) \log \frac{p(\theta, s)}{p(\theta) p(s)} d\theta ds = \mathbb{E} \left[ \log \frac{p(\theta \mid S)}{p(\theta)} \right]. \quad (1.252)$$

In terms of the KL divergence between prior and posterior,

$$I(\theta; S) = \mathbb{E}_S [\text{KL}(p(\theta \mid S) \parallel p(\theta))]. \quad (1.253)$$

So,  $I(\theta; S)$  also measures the average distance between posterior and prior.

Information Loss measures how much information is lost when replacing  $y$  with  $S(y)$ . Mathematically,

$$\Delta = I(\theta; y) - I(\theta; S(y)) = H(\theta \mid S(y)) - H(\theta \mid y). \quad (1.254)$$

Fisher Information measures how sensitive the likelihood  $p(S(y) | \theta)$  and  $p(y | \theta)$  are to  $\theta$ , respectively. Mathematically,

$$\mathcal{I}_{S(y)}(\theta) \leq \mathcal{I}_y(\theta). \quad (1.255)$$

Equivalently,

$$\mathbb{E} \left[ \left( \frac{\partial}{\partial \theta} \log p(S(y) | \theta) \right)^2 \right] \leq \mathbb{E} \left[ \left( \frac{\partial}{\partial \theta} \log p(y | \theta) \right)^2 \right]. \quad (1.256)$$

So, the loss of Fisher Information also measures how much information is lost when replacing  $y$  with  $S(y)$ , and sufficiency means the inequality becomes equality. Mathematically,

$$\Delta = \mathbb{E} \left[ \left( \frac{\partial}{\partial \theta} \log p(S(y) | \theta) \right)^2 \right] - \mathbb{E} \left[ \left( \frac{\partial}{\partial \theta} \log p(y | \theta) \right)^2 \right]. \quad (1.257)$$

The posterior variance reduction measures how much noise is removed given the observed data. This compares the variance of the prior  $p(\theta)$  and the estimated posterior  $p(\theta | S(y))$  in ABC. Note that the true posterior is  $p(\theta | y)$ . Mathematically,

$$\Delta = \text{Var}(\theta | S(y)) - \text{Var}(\theta). \quad (1.258)$$

## 1.9 Dimensionality and Interpretability of Summary Statistics

In this project, dimensionality addresses whether the chosen statistics are sufficiently low-dimensional to enable efficient processing and analysis, and interpretability addresses whether they can be meaningfully mapped to biological components and processes.

### 1.9.1 Dimensionality of Summary Statistics

The dimensionality of the summary statistics is defined as

$$d = \dim(S(x)), \quad S(x) \in \mathbb{R}^d. \quad (1.259)$$

The performance of ABC is highly sensitive to the dimensionality  $d$ . As  $d$  increases, the volume of the space  $\mathbb{R}^d$  grows exponentially, making it increasingly unlikely that simulated summaries  $S(x)$  fall within an  $\epsilon$ -ball around  $S(x_{\text{obs}})$ . Consequently, the acceptance probability decreases exponentially with  $d$ , leading to severe computational inefficiency [7].

The choice of dimensionality introduces a fundamental tradeoff:

1. Low-dimensional summaries: Higher acceptance rates but potential loss of information, resulting in biased posterior approximations.
2. High-dimensional summaries: Retain more information about the data and parameters, but lead to lower acceptance rates and higher variance in estimates.

Ideally, one seeks summary statistics  $S(x)$  that are sufficient for  $\theta$ , i.e.,

$$p(\theta \mid x) = p(\theta \mid S(x)). \quad (1.260)$$

In this case, no information about  $\theta$  is lost by compressing the data. However, in most practical applications, sufficient statistics are either unknown or high-dimensional, making them impractical for ABC. As a result, practitioners typically construct low-dimensional statistics that are informative but not strictly sufficient [8].

The dimensionality of summary statistics governs a tradeoff between statistical efficiency and computational feasibility: low-dimensional summaries improve acceptance rates but may introduce bias, while high-dimensional summaries improve accuracy but reduce efficiency.

### 1.9.2 Interpretability of Summary Statistics

Interpretability refers to the extent to which the components of  $S(x)$  have clear and meaningful interpretations in terms of the underlying data-generating process.

Common examples include:

1. Mean: measures central tendency,
2. Variance: measures dispersion,
3. Autocorrelation: captures temporal dependence,

#### 4. Spectral features: characterize oscillatory behavior.

Interpretable summaries allow practitioners to identify which aspects of the data are most informative for inferring parameters. When ABC produces poor approximations, interpretable statistics enable the diagnosis of missing or insufficient features.

In scientific applications, such as biology or neuroscience, interpretable summaries provide explanatory insights into the mechanisms underlying the data, beyond mere predictive performance. Interpretability connects statistical inference to domain knowledge, enabling a meaningful interpretation of inferred parameters and model behavior.

### 1.9.3 Dimensionality of Machine Learning-based Summary Statistics

Recent advances in ABC replace handcrafted summary statistics with learned representations, typically parameterized by neural networks. In ABC-SMC, this yields a framework in which both the summary statistics and, in some cases, the distance function are learned from data.

Let  $x \sim p(x \mid \theta)$  denote the simulated data. ML-based summary statistics take the form

$$S_\phi(x) = f_\phi(x), \quad (1.261)$$

where  $f_\phi$  is a neural network with parameters  $\phi$ . In addition, one may define a learned distance function

$$\rho_\psi(S_\phi(x), S_\phi(y)), \quad (1.262)$$

where  $\psi$  parameterizes the distance metric, and  $y$  indicates observations.

Unlike conventional ABC, in which the dimensionality of summary statistics is manually chosen and constrained to be small, ML-based summary statistics allow flexible dimensionality choices. They can be low-dimensional (e.g.,  $d \in [5, 20]$ ), moderate-dimensional (e.g.,  $d \in [32, 128]$ ), or high-dimensional feature embeddings (e.g.,  $d > 128$ ) in deep architectures.

The dimensionality  $d = \dim(S_\phi(x))$  is thus a design parameter rather than a theoretical constraint.

In conventional ABC, summary statistics are ideally chosen to be sufficient. However, for complex models with intractable likelihoods, sufficient statistics are generally unknown. Neural approaches instead aim to learn representations that are informative about the parameter  $\theta$

$$S_\phi(x) \approx S^*(x), \quad (1.263)$$

where  $S^*(x)$  denotes a (unknown) sufficient statistic.

This objective is typically achieved indirectly through optimization criteria (i.e., specified loss function) such as:

1. Regression-based objectives: learning  $S_\phi(x)$  to predict  $\theta$  or  $\mathbb{E}[\theta | x]$ ,
2. Mutual information maximization: maximizing  $I(S_\phi(x); \theta)$ ,
3. Classification-based objectives: distinguishing between parameter-data pairs.

These approaches encourage  $S_\phi(x)$  to retain information about  $\theta$  without explicitly enforcing sufficiency [17, 28].

Although ML-based summaries may be high-dimensional, their intrinsic dimensionality can be significantly lower. That is, even if

$$S_\phi(x) \in \mathbb{R}^d, \quad (1.264)$$

the effective information content may lie on a lower-dimensional manifold. That means, the actual variability of the data may lie on a much smaller structure which is

$$S_\phi(x) \in \mathcal{M} \subset \mathbb{R}^d, \quad \dim(\mathcal{M}) = k \ll d, \quad (1.265)$$

where  $\mathcal{M}$  is a low-dimensional manifold;  $k$  is the intrinsic dimension;  $d$  is the ambient dimension.

This mitigates, to some extent, the curse of dimensionality that affects conventional ABC methods.

In ABC-SMC with learned distance functions,

$$\rho_\psi(S_\phi(x), S_\phi(y)). \quad (1.266)$$

That means the geometry of the summary space is adapted during training. As a result, distances become task-dependent and data-driven, so that relevant directions in the summary space are emphasized, and irrelevant variability can be suppressed.

Consequently, higher-dimensional summaries become more tractable compared to conventional ABC, since both representation and distance are jointly optimized.

ML-based summary statistics relax the strict low-dimensionality requirement of conventional ABC by jointly learning representation and geometry, thereby improving statistical efficiency while maintaining computational tractability.

### 1.9.4 Interpretability of Machine Learning-based Summary Statistics

Unlike conventional summary statistics that are handcrafted, ML-based summaries,

$$S_\phi(x) = f_\phi(x), \quad (1.267)$$

are typically latent feature representations.

These features lack direct semantic meaning, so they are not human-interpretable and may exhibit complex and entangled dependencies.

Despite their lack of interpretability, ML-based summaries are optimized to capture information relevant for inference. That means

$$S_\phi(x) \approx \text{features predictive of } \theta. \quad (1.268)$$

Thus, they are task-optimal, in the sense of maximizing predictive information about  $\theta$ , but not necessarily interpretable in terms of domain-specific quantities.

Interpretability can be partially recovered through post-hoc analysis techniques, including feature importance measures, sensitivity analyses, dimensionality reduction, and visualizations.

However, such interpretations are indirect and do not correspond to explicit, predefined statistical quantities.

ML-based summary statistics trade interpretability for adaptivity. They provide more informative and flexible representations for inference at the cost of losing direct semantic meaning. In this project, the regression provides summary statistics for the calibration parameters gamma and alpha directly, so they have specific biological interpretations. Representation learning does not provide interpretable meanings for summary statistics, as the summary statistics are latent embeddings generated by the trained model.

## 1.10 Problems and Objectives

The Semi-automatic ABC pipeline requires a single ML or DL model to learn the pattern and select informative summary statistics. It was originally a regression model for the task, and then machine learning or even deep learning models were applied to improve training performance. However, none of these models are particularly helpful for this single-cell growth project, and there are two main problems to consider:

**Problem 1.** The real experiments are handled by microscopy, and images are collected sequentially over time, as are the simulations. All images constitute spatial-temporal data in a time series, and each image differs across time points. Cells grow and divide, so the number of cells and their locations vary over time within a single microcolony. Then the number of inputs in an ML or DL model could not be fixed. Many conventional neural networks require a fixed number of inputs. GNNs do not require a fixed number of inputs, making them well-suited to images at different time points.

**Problem 2.** CNNs are a common DL model for processing, segmenting, and tracking imaging data, but they rely on Euclidean distance, which makes training computationally expensive. In this single-cell growth model, the biophysical behaviors that are used are independent of the Euclidean distance between cells. ABC requires high computation, and cell-growth model simulations are usually quite slow and computationally demanding, so the Euclidean distance might be a significant issue in this case. However, a GNN does not compute Euclidean distances; instead, it is primarily composed of edges and nodes that represent relationships among cells.

**Objective.** The objective of this project is to replace manually selected conventional summary statistics with GNN-based summary statistics in the ABC pipeline for ABM parameter inference. We also need to evaluate the informativeness of GNN-based summary statistics and compare the posterior inference results between the conventional and GNN-based ABC pipelines. Eventually, we intend to use these informative GNN-based summary statistics in our ABC pipeline to infer the true parameter posteriors, according to both the ground-truth baseline and observations from real experiments.

# Chapter 2

## Methods

### 2.1 Datasets and Features

#### 2.1.1 Data Formats

The experimental data were originally acquired with microscopes and subsequently processed using `ImageJ` and `CellProfiler`. This generates experimental csv files as input to `TrackRefiner`. `TrackRefiner` refines them and re-formats the data and features so that both simulated and experimental csv files share the same features and use the same data format. Then `PyG` (or `PyTorch Geometric`) processes all csv files and converts them into graph data for subsequent training and prediction steps, but still in csv format. Figure 10 shows how the simulated and experimental data are handled as two data formats at different stages. All processing steps do not change the information already present; instead, they refine and adjust the data formats to preserve the information in the original source from which the data are generated.

The simulated data are generated by running `CellModeller`, the chosen ABM simulation package, to record the biophysical features of microcolonies as they grow over time [48]. Each pickle file contains a large dictionary with keys `cellStates`, `stepNum`, `lineage`, `moduleStr`, and `moduleName` that record the simulation’s progress over time. The output processing script processes the simulated pickle files and converts them into ready data for input to `TrackRefiner`. `TrackRefiner` comprises 2 parts: one to refine the imaging data, and another to reformat the data and features. `TrackRefiner` takes the output of the processing script and reformats the data and features. One of the purposes of `TrackRefiner`

is to produce both simulated and experimental data in the same format. There is no refinement part for the simulated data, but both parts would be applied to the experimental data.

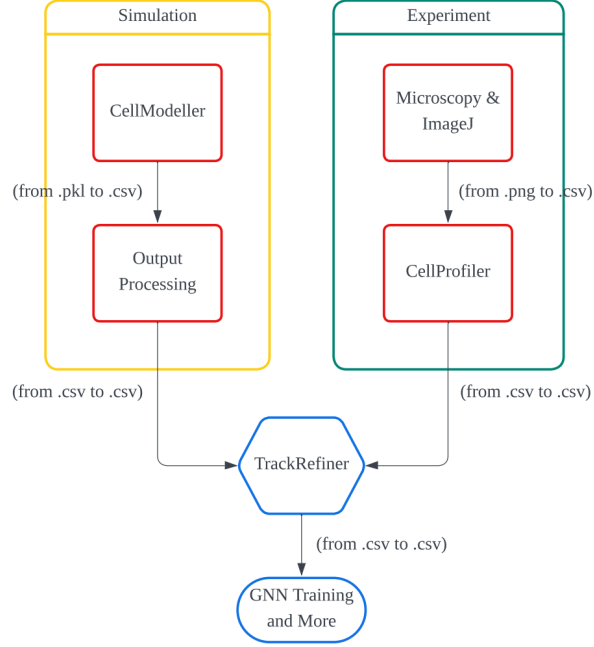


Figure 2.1: Different data formats in different stages of the project.

### 2.1.2 Parameter Value Selection for Training Data

To run simulations in `CellModeller`, the observable ABM parameters are fixed, whereas the unobservable parameters  $\gamma$  and  $\alpha$  must be assigned specific values, which typically vary across the calibration. Different  $\gamma$  and  $\alpha$  values generate different data because of themselves. Even identical  $\gamma$  and  $\alpha$  values can yield different data due to the stochasticity of observable ABM parameters. The purpose of training a GNN model is to make it learn the patterns from different  $\gamma$  and  $\alpha$  values in a particular range of interest. Different ways of selecting  $\gamma$  and  $\alpha$  values lead to different training performance.

This project primarily includes two learning tasks: representation learning and regression. The details of both learning tasks will be provided later, but here we present different ways of parameter selection for the two tasks.

Here, the representational learning is defined by its triplet loss function, so it operates in a manner similar to classification. For this task, 10 gamma and 10 alpha values are typically selected. There is no reason to fix specific parameter values, provided the range encompasses experimental samples, and the selected values are biologically representative. The reasonable range of gamma is usually 1 to 1000, and the reasonable alpha range is 0.1 to 100. One gamma value pairs with one alpha value to form a label that functions the same as a class in classification tasks. Each pair of parameters yields multiple different simulations, which can help the GNN model capture patterns. In the representation learning part of this project, the selected gamma and alpha values are

| $\gamma$  | $\alpha$ |
|-----------|----------|
| 1.99779   | 1.47881  |
| 3.79126   | 2.17266  |
| 6.96415   | 3.37139  |
| 13.05505  | 5.05364  |
| 23.31662  | 7.75259  |
| 42.81914  | 12.11043 |
| 77.91484  | 18.13678 |
| 149.97132 | 28.24211 |
| 275.89982 | 43.40776 |
| 520.30146 | 64.54524 |

Table 2.1: Values of  $\gamma$  and  $\alpha$

For the regression task, 1000 gamma and 1000 alpha values were randomly sampled from the given log-uniform distributions. Then each gamma value is paired with an alpha value to form a multivariate label for each sample, yielding 1000 samples in total. A log-uniform distribution is appropriate here and would be a good choice given the limited prior knowledge of the gamma and alpha parameters from the experimental data, as it is more biologically representative. For both representation learning and regression tasks, gamma and alpha values are selected and spaced regularly, but are more right-skewed because prior knowledge in the ABC indicates that the logarithm of gamma and alpha is uniformly distributed. Figure 11 shows the distributions of the gamma and alpha values.

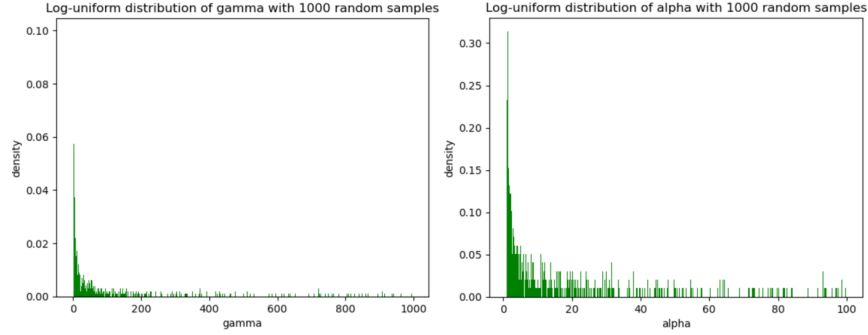


Figure 2.2: Proposed distributions for parameter selection with different numbers of samples.

The logarithm-uniform distribution of parameter  $X$  means that the logarithm of parameter  $S$  should be uniformly distributed, so we set

$$S \sim \text{Uniform}(a, b), \quad (2.1)$$

and define

$$X = 10^S, \quad (2.2)$$

such that equivalently

$$S = \log_{10}(X). \quad (2.3)$$

More specifically,

$$\gamma \sim \text{Loguniform}(1, 1000), \quad (2.4)$$

which means

$$\log_{10}(\gamma) \sim \text{Uniform}(\log_{10}(1), \log_{10}(1000)), \quad (2.5)$$

and

$$\alpha \sim \text{Loguniform}(0.1, 100), \quad (2.6)$$

which means

$$\log_{10}(\alpha) \sim \text{Uniform}(\log_{10}(0.1), \log_{10}(100)). \quad (2.7)$$

The logarithmic base of 10 is replaced with the logarithmic base of  $e$ , and the lower bound is arbitrarily switched from 0 to 1 for computational reasons.

In the ABC method, prior knowledge of the experimental data is not helpful for predicting the parameters, so a uniform distribution is assumed on the logarithms of the gamma and alpha parameters. It would be preferable to select parameter values for GNN training that are close to the prior distribution, since this ensures a sufficiently wide range to cover the predictions from the experimental data.

### 2.1.3 Data Cleaning and Preprocessing

The cleaning and preprocessing steps are as follows. Note that there is no need to remove highly multi-correlated features (across graphs) because all the activation functions in the neural network are non-linear. Note that the empty value `NaN` should be replaced within each graph, since each graph has an independent mean value from the other graphs.

1. Fill in `NaN` with mean (within each graph)
2. Convert data to all floats (within each graph)
3. Flatten vectors (within each graph)
4. Convert T/F to 1/0 (within each graph)
5. Remove constant features (across graphs)

Note that more descriptions about the data and features can be found in [Appendix A](#).

### 2.1.4 Feature Importance Ranking

Feature importance ranking is a machine learning method that orders input features by their contribution to model predictions.

Let  $X = (x_1, x_2, \dots, x_d)$  denote the set of features. A feature importance ranking assigns a score  $I_j$  to each feature  $x_j$ , i.e.,

$$I_j, \quad j = 1, \dots, d. \quad (2.8)$$

The features are then ordered according to their importance:

$$x_{(1)} \succ x_{(2)} \succ \dots \succ x_{(d)}, \quad (2.9)$$

where  $x_{(1)}$  denotes the most important feature.

In this work, we employ a Graph Attention Network (GAT)-based attention-weighted feature importance method tailored to our model.

To analyze feature importance in GNNs, an attention-weighted feature-scoring method was proposed that uses the learned attention coefficients from a GAT layer.

Given node features  $h_i \in \mathbb{R}^d$ , a GAT layer computes

$$h'_i = \sigma \left( \sum_{j \in \mathcal{N}(i)} \alpha_{ij} W h_j \right), \quad (2.10)$$

where  $\alpha_{ij}$  are attention coefficients:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(a^\top [W h_i \parallel W h_j]))}{\sum_{k \in \mathcal{N}(i)} \exp(\cdot)}. \quad (2.11)$$

Let  $e = (s \rightarrow t)$  denote an edge with source node  $s$  and target node  $t$ . For each feature  $f$ , we define the feature difference

$$d_{e,f} = |x_{s,f} - x_{t,f}|. \quad (2.12)$$

Let  $\alpha_e$  denote the attention weight associated with edge  $e$ , averaged between the attention heads. The feature importance score at a given layer is computed as

$$I_f^{(\ell)} = \sum_{e \in \mathcal{E}} \alpha_e^{(\ell)} \cdot d_{e,f}. \quad (2.13)$$

Aggregating across layers  $\mathcal{L}$ , we obtain

$$I_f = \sum_{\ell \in \mathcal{L}} I_f^{(\ell)}. \quad (2.14)$$

To improve robustness, we average the scores across mini-batches

$$I_f^{\text{avg}} = \frac{1}{B} \sum_{b=1}^B I_f^{(b)}, \quad (2.15)$$

and normalize:

$$\tilde{I}_f = \frac{I_f^{\text{avg}}}{\sum_{f'} I_{f'}^{\text{avg}}}. \quad (2.16)$$

This formulation assigns greater importance to features that exhibit large differences between connected nodes, thereby capturing how strongly feature variations contribute to message passing in the graph.

This approach leverages the GAT model's learned attention structure and is naturally aligned with the message-passing mechanism. However, attention weights do not necessarily provide a faithful explanation of model predictions, and thus, the resulting importance scores should be interpreted with caution.

## 2.2 Simulated-experimental Data

Real experimental data are defined as samples from real experiments. The simulated experimental data are defined as the samples from simulations. As described in [subsection 2.5.3](#), to generate the simulated experimental data, the observable ABM parameters are still fixed as in the previous setup. The calibration parameters  $\gamma$  and  $\alpha$  are randomly selected from uniform distributions as well, but the key point is that the simulated experimental data should be a different group of data from the simulated data for GNN training. The number of samples is the same as that in real experiments. However, there is no need to select the calibration parameters from a uniform distribution as described in [subsection 2.5.3](#). Here, this helps improve GNN performance on simulated-experimental data because both distributions are uniform. It is a way to use fewer training samples to achieve better performance, but they do not need to be drawn from the same distribution because of the generalization properties of GNN models. However, more training samples may be needed to maintain high enough performance.

In the ABC pipeline, we are supposed to use real experimental data to evaluate the performance of trained models and inferred parameter distributions, but we replace them with simulated data. The real experimental data are unlabeled, and therefore the associated parameters are unknown. In contrast, the simulated experimental data are generated from independent simulations with fixed parameter values. Within the ABC pipeline, these simulated datasets are treated as real experimental data. The fixed, known parameters are modeled as a delta function, serving as a baseline for comparing results across different strategies.

## 2.3 Pipeline Overview

In this project, we used ABC-SMC as the inference pipeline, as described in [section 1.6](#). It is an adaptive weighted ABC rejection algorithm, one of the most popular variants of ABC.

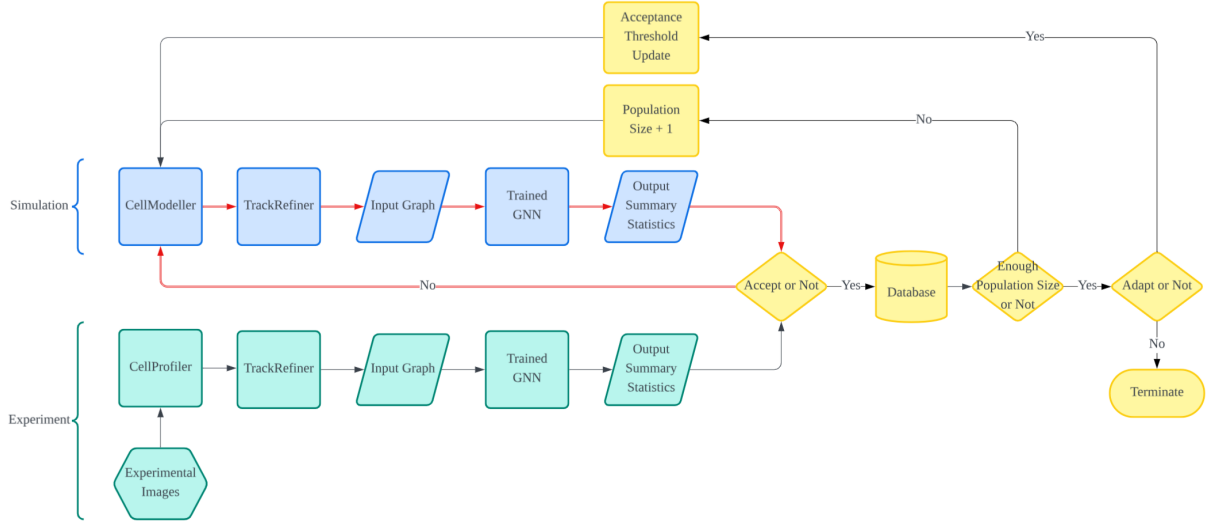


Figure 2.3: A pipeline overview flowchart. The arrows highlighted in red denote a simulator circulation or a circulation of a simulator set with a fixed pair of parameters (i.e., gamma and alpha). From left to right, the first decision gate compares simulated summary statistics with experimental summary statistics; the second decision gate determines whether the population size is enough to finish the current generation; the third decision gate determines whether the current generation adapts to the next one with a lower acceptance threshold in the distance metric.

The goal of this project is to replace conventional summary statistics  $S$  with GNN-based summary statistics (a type of ML-based summary statistics as described in [section 1.7](#)). We define a GNN-based summary statistic

$$S_{\phi(GNN)} = S_{\phi} : \mathcal{Y} \rightarrow \mathbb{R}^k, \quad (2.17)$$

which maps data (either observed data  $y$  or simulated data  $x$ ) to a  $k$ -dimensional feature vector. Note that  $\mathcal{Y}$  is the input space and  $\mathbb{R}^k$  is the output space.

All ABC terminations are set to occur when the adaptive acceptance threshold epsilon reaches the minimum value of 0.1 after a few generations, and this threshold can be customized. There are no other limitations to stop the pipeline other than this. Sometimes a high epsilon value arises because a summary statistic is close to 0, since these results are obtained by running ABC with respect to the relative distance. The min-max normalization is applied to all summary statistics to prevent values near 0. The gamma value is fixed

at 200, and the alpha value is fixed at 35 for all strategies. These figures are logarithmic scale, base 10 of parameters, namely

$$\gamma = 200, \quad (2.18)$$

$$\alpha = 35, \quad (2.19)$$

$$\ln(\gamma) \approx 2.301, \quad (2.20)$$

$$\ln(\alpha) \approx 1.544. \quad (2.21)$$

Each observation is represented as a multi-relational graph that is described in [subsection 1.3.3](#)

$$G = (V, E^{(c)}, E^{(l)}, X), \quad (2.22)$$

where  $V$  is the set of nodes,  $E^{(c)}$  and  $E^{(l)}$  denote contact and lineage edges respectively, and  $X \in \mathbb{R}^{|V| \times d}$  contains node-level features.

The goal is not to replace these features, but to learn a mapping from graph-structured data to low-dimensional summary statistics suitable for inference.

We employ a multi-relational GNN to encode both node features and edge structure. Let  $h_i^{(0)} = x_i$  denote the initial node features. As described in [Equation 1.136](#), the basic hidden representations are updated as

$$\mathbf{h}_u^{(k)} = \sigma \left( \mathbf{W}_{\text{self}}^{(k)} \mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{contact}}^{(k)} \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(k-1)} + \mathbf{W}_{\text{lineage}}^{(k)} \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)} \right). \quad (2.23)$$

So,

$$\mathbf{h}_u^{(k)} = \sigma \left( \mathbf{W}_{\text{self}}^{(k)} \mathbf{h}_u^{(k-1)} + \sum_{r \in \{c, l\}} \sum_{v \in N_r(u)} \mathbf{W}_r^{(k)} \mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)} \right), \quad (2.24)$$

where  $c$  indicates contact neighbors;  $l$  indicates lineage neighbors;  $r$  indicates the edge type index.

This architecture allows the model to jointly incorporate node-level features  $X$ , contact interactions, and lineage relationships.

After  $L$  layers, node embeddings  $\{h_i^{(L)}\}_{i \in V}$  are aggregated into a graph-level representation

$$S_\phi(G) = \text{READOUT} \left( \{h_i^{(L)}\}_{i \in V} \right). \quad (2.25)$$

In this work, we use mean pooling

$$S_\phi(G) = \frac{1}{|V|} \sum_{i \in V} h_i^{(L)}. \quad (2.26)$$

The function  $S_\phi(G) \in \mathbb{R}^k$  serves as a GNN-based summary statistic, combining node features and relational structure into a compact representation.

There are two main comments on this ABC inference pipeline. One is that the **CellModeller** simulation module setup is assumed to be sufficiently close to the real experiments, except for the calibration parameters gamma and alpha. In this case, a good evaluation result on simulated-experimental data means a good result on real-experimental data, because there is no mismatch between them. Another issue is that in real experiments, cells grow but do not die because each experimental sample is observed for only 3 minutes, whereas the simulation also accounts for cell growth but does not model cell death. There are some biological methods to count cell death as well, but the calibration parameters gamma and alpha are assumed to be independent of cell death. However, there is no difference, although cells also die during the process, because gamma and alpha are only associated with a limited set of biophysical behaviors in a cell growth model.

For each micro-colony, experimental images are acquired over time, so the simulated graphs are also time series. This project proposes four strategies for training the GNN model. Two of them use sequential graphs over time because they employ LSTM to model temporal dependencies. The other two use the cumulated graphs, and node-level features can capture this temporal information. All details are provided in the following sections.

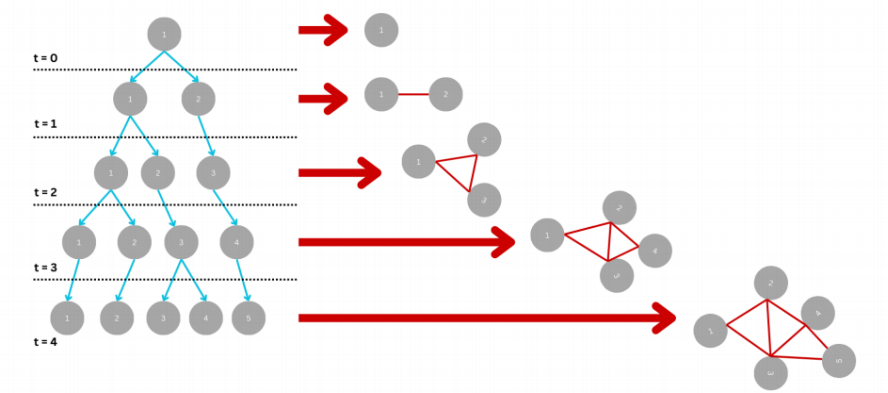


Figure 2.4: Cell Lineage and Neighbor [13]. The left side shows how the cell divides; notice that not every cell divides at each time step. At each time step, each node will have a different ID. The right panel shows the layout of the cell at that time step, along with the contact information for the cells. This figure is from Sushma Dhamodharan, one of the previous master's students in our research group.

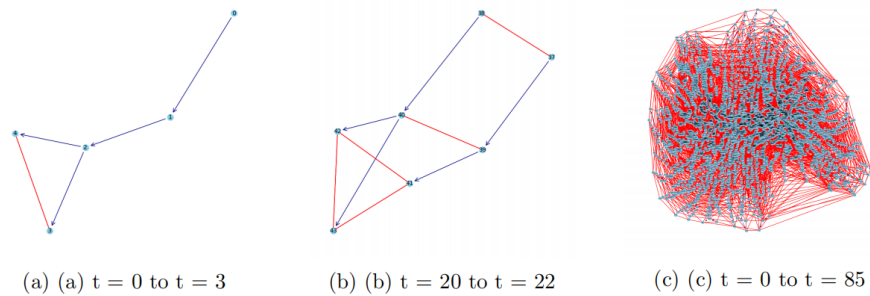


Figure 2.5: Graph representation of colony for various intervals [13]. Red lines are contact edges, and blue arrows are directed lineage edges. This figure is from Sushma Dhamodharan, one of the previous master's students in our research group.

## 2.4 Proposed Graph Neural Network-based Architectures

### 2.4.1 Training Setup

The following training setups are shared among the GNN-based strategies, which will be explained in detail in the next couple of sections.

1. Data size: We simulated 1000 samples for training.
2. Activation function: We selected ELU as the activation function in GNNs.
3. Data type: Graphs (nodes and edges) were loaded as `torch_geometric.data.Data`.
4. Data split: 70% training, 15% validation, 15% testing.
5. Batching: Batch size = 50 graphs for training, validation and testing.
6. Optimizer: **AdamW** where learning rate =  $10^{-3}$  and weight decay =  $10^{-4}$ .
7. Training schedule: The model was trained for up to 500 epochs.
8. Hardware: **a-100** and **p-100** GPUs and CPUs.
9. Model persistence: Trained models were saved in PyTorch `state_dict` format.

All GNN-based strategies applied Graph Attention Network (GAT) layers, or GAT layers with LSTM layers. GAT and GCN are two basic GNN variants, and GAT works better than GCN in complex network systems. We aim to be clear about our intentions and to use simple models to illustrate our ideas. That means that if a simple model works, there are many more sophisticated models that we can try afterward.

To prevent overfitting during training, early stopping was employed. The validation loss was monitored after each epoch, and the model parameters corresponding to the lowest validation loss were stored. Here, early stopping was applied based on validation loss with a patience of 20 epochs. The model parameters corresponding to the lowest validation loss were retained as the final trained model.

At the early stage of this project, the parameters are the logarithms of gamma and alpha in base 10, but converted to a linear scale to run **CellModeller** simulations. Once

simulations are performed, the parameter scales are normalized and standardized for training. Afterward, the trained GNN model generates a linear scale of parameters by learning this pattern. Finally, the log of parameters in base 10 is converted back to the log scale. The base-10 logarithm has recently been replaced by the base-e logarithm in this project.

### 2.4.2 First Strategy: Graph Neural Network Regression in Approximate Bayesian Computation

The first strategy is a graph-level regression task in which the model predicts a target value for each input graph. Note that all the details of the following formulations can be referred to [subsection 1.3.5](#). To understand [Figure 2.6](#), we consider a heterogeneous graph

$$G = (V, E_c, E_l), \quad (2.27)$$

where  $E_c$  indicates the undirected contact edges,  $E_l$  indicates the directed lineage edges,  $V$  indicates the set of vertices (nodes).

We define the neighborhoods of any arbitrary node  $i \in V$  through contact edges and lineage edges as

$$\mathcal{N}_l(i) = \{j \mid (j \rightarrow i) \in E_l\}, \quad \mathcal{N}_c(i) = \{j \mid (i, j) \in E_c\}, \quad (2.28)$$

where  $j \in V$  is any arbitrary neighbor node of  $i$ .

Note that, in the architectures, everything up to the graph embeddings belongs to the GNN-based encoder, and everything after it belongs to the decoder, so each strategy has its specified encoder and decoder.

In GAT layer 1 with lineage edges,

$$z_i^{(1)} = W_l^{(1)} h_i^{(0)} + b_l^{(1)}, \quad (2.29)$$

$$e_{ij}^{(1)} = \text{LeakyReLU} \left( a_l^{(1)\top} \left[ z_i^{(1)} \parallel z_j^{(1)} \right] \right), \quad j \in \mathcal{N}_l(i), \quad (2.30)$$

$$\alpha_{ij}^{(1)} = \frac{\exp(e_{ij}^{(1)})}{\sum_{m \in \mathcal{N}_l(i)} \exp(e_{im}^{(1)})}, \quad (2.31)$$

$$h_i^{(1)} = \text{ELU} \left( \sum_{j \in \mathcal{N}_l(i)} \alpha_{ij}^{(1)} z_j^{(1)} \right). \quad (2.32)$$

In GAT layer 2 with lineage edges,

$$z_i^{(2)} = W_l^{(2)} h_i^{(1)} + b_l^{(2)}, \quad (2.33)$$

The same way to get the attention scores and weights as [Equation 2.30](#) and [Equation 2.31](#), so

$$h_i^{(2)} = \text{ELU} \left( \sum_{j \in \mathcal{N}_l(i)} \alpha_{ij}^{(2)} z_j^{(2)} \right). \quad (2.34)$$

GAT layers 1 and 2 are used to propagate along lineage edges and capture ancestry and temporal dependencies.

In GAT layer 3 with contact edges,

$$z_i^{(3)} = W_c^{(3)} h_i^{(2)}. \quad (2.35)$$

Same way to get the attention scores and weights as [Equation 2.30](#) and [Equation 2.31](#), so

$$h_i^{(3)} = \text{ELU} \left( \sum_{j \in \mathcal{N}_c(i)} \alpha_{ij}^{(3)} z_j^{(3)} \right). \quad (2.36)$$

In GAT layer 4 with contact edges,

$$z_i^{(4)} = W_c^{(4)} h_i^{(3)}. \quad (2.37)$$

GAT layers 3 and 4 are used to propagate along contact edges and capture spatial interactions between nodes.

Same way to get the attention scores and weights as [Equation 2.30](#) and [Equation 2.31](#), so

$$h_i^{(4)} = \text{ELU} \left( \sum_{j \in \mathcal{N}_c(i)} \alpha_{ij}^{(4)} z_j^{(4)} \right). \quad (2.38)$$

To aggregate node-level information into a graph embedding,

$$h_G = \frac{1}{N} \sum_{i=1}^N h_i^{(4)}. \quad (2.39)$$

To reduce the graph embedding dimension and produce the final regression output,

$$\hat{y} = W_{\text{out}} h_G + b_{\text{out}}. \quad (2.40)$$

Repeat the above steps for all graph samples to get a prediction set  $\{\hat{y}_1, \hat{y}_2, \hat{y}_3, \dots, \hat{y}_N\}$  corresponding to the target set  $\{y_1, y_2, y_3, \dots, y_N\}$  where  $N = 50$  indicates the number of samples in a batch (or the number of epoch if no batching is applied). To determine the MSE,

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2. \quad (2.41)$$

Figure 2.6 is used to train the GNN model  $S_\phi(G)$  with respect to the epoch-wise Equation 2.41, which maps the input graph to the GNN-based summary statistics as described in section 2.3. Note that GNN-based summary statistics are applied to the ABC pipeline for parameter inference as described in Figure 2.3. So, given experimental graph observations  $\{G_{\text{exp}}^{(m)}\}_{m=1}^M$ , we have

$$s_{\text{exp}}^{(m)} = S_\phi(G_{\text{exp}}^{(m)}), \quad (2.42)$$

and define the empirical mean summary

$$\bar{s}_{\text{exp}} = \frac{1}{M} \sum_{m=1}^M s_{\text{exp}}^{(m)}. \quad (2.43)$$

For simulated graphs  $G \sim p(\cdot \mid \theta)$ , we compute

$$s_{\text{sim}}(\theta) = S_\phi(G). \quad (2.44)$$

The discrepancy used in ABC is defined as

$$d(\theta) = \|S_\phi(G) - \bar{s}_{\text{exp}}\|_1. \quad (2.45)$$

Or

$$d(\theta) = D_{\text{KL}}(P_{\text{exp}}(S_\phi(G)), P_{\text{sim}}(S_\phi(G) \mid \theta)). \quad (2.46)$$

Suppose the dynamical graph is captured in parts over time. Subsequently, sequential graphs were used to represent the dynamics of the growing colony. The first strategy treats a sequence of graphs as a cumulative static graph (i.e., stacks all graphs into a single graph), thereby converting temporal information into an additional edge type, composed of lineage edges between nodes. Physical connections are represented by contact edges that connect each target node to all of its neighbors. Note that contact edges are undirected and lineage edges are directed here.

The configured dimensions are:

1. Node feature dimension:  $d_{\text{in}} = 31$ ,
2. Number of predicted parameters:  $d_{\text{out}} = 2$ ,
3. GNN hidden dimension:  $d_{\text{hidden}} = 4d_{\text{in}}$ ,

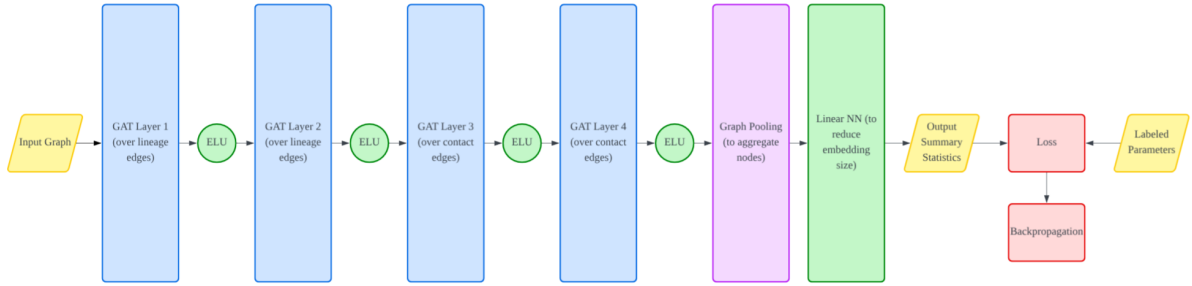


Figure 2.6: The architecture of the first strategy: GNN regressor in the ABC. This architecture explains what’s inside the trained GNN model, as shown in Figure 2.3, which is the pipeline overview flowchart in section 2.3. The output summary statistics are the parameter gamma and alpha predicted by the GNN regressor. The labeled targets are the parameters gamma and alpha as well.

Note that the scales of the gamma and alpha parameters vary throughout the pipeline. This is mainly because of 2 reasons. One is that **CellModeller** scales exponentially to generate simulations. The other is that the node features and the graph targets (i.e., the gamma and alpha parameters in regression) must be normalized for training. Summary statistics (i.e., graph representations in embedding learning) also need to be normalized, as doing so helps prevent any element of the embedding vector from being too close to zero, which can cause the acceptance threshold in ABC to be exceeded. The relative distance

between summary statistics does not require normalization because it is computed relative to each parameter. The distribution-based distance between the summary statistics, measured by the KL divergence, should use normalized summary statistics, as it is not a relative metric. The standard normalization is applied to the parameter and node features, and the min-max normalization is applied to the graph embeddings as summary statistics in the ABC. This is based on parallel test results, and the best-performing ones would be selected. Note that the scale transformations in the ABC inference part should be kept the same as in the GNN training part, because the trained GNN model is applied directly to the ABC inference.

| Stages              | $\theta$ transformation                                                                                 |
|---------------------|---------------------------------------------------------------------------------------------------------|
| Beginning           | $\log_{10}(\theta)$                                                                                     |
| <b>CellModeller</b> | $\theta$                                                                                                |
| GNN begins          | $z = \frac{\log_{10}(\theta) - \mu}{\sigma}$                                                            |
| GNN ends            | $\log_{10}(\theta)$                                                                                     |
| ABC begins          | $\frac{\log_{10}(\theta) - \min(\log_{10}(\theta))}{\max(\log_{10}(\theta)) - \min(\log_{10}(\theta))}$ |
| ABC ends            | $\log_{10}(\theta)$                                                                                     |

Table 2.2: Transformations applied to the parameter  $\theta$  during different stages for the regression tasks (i.e., the first and the second strategy) with relative distance of summary statistics. Note that the summary statistics may also require normalization if the relative distance is replaced with a distribution-based distance and the KL divergence is used.

| Stages     | Node features $X$ transformation |
|------------|----------------------------------|
| GNN begins | $Z = \frac{X - \mu}{\sigma}$     |
| GNN ends   | $X$                              |

Table 2.3: Transformations applied to the node features  $X$  during different stages for the regression tasks (i.e., the first and the second strategy). Note that  $X$  means the node feature matrix that includes each node feature  $x$ .

| Stages     | Summary statistics $S_\phi(x)$ transformation                                         |
|------------|---------------------------------------------------------------------------------------|
| ABC begins | $\tilde{S}_\phi(x) = \frac{S_\phi(x) - S_\phi^{\min}}{S_\phi^{\max} - S_\phi^{\min}}$ |
| ABC ends   | $S_\phi(x)$                                                                           |

Table 2.4: Transformations applied to the summary statistics  $S_\phi(x)$  during different stages for the regression tasks (i.e., the first and the second strategy). Note that the summary statistics may also require normalization if the relative distance is replaced with a distribution-based distance and the KL divergence is used.

### 2.4.3 Second Strategy: Graph Neural Network Regression in Approximate Bayesian Computation with Long Short-term Memory

The second strategy is also a graph-level regression task in which the model predicts a target value for each input graph. Instead of using lineage edges, it applies LSTM layers to capture temporal information in the growing population dynamics. Note that all the details of the following formulations can be referred to [subsection 1.3.2](#) and [subsection 1.3.5](#). To understand [Figure 2.7](#), we consider a sequence of graphs

$$\mathcal{G} = \{G_1, G_2, \dots, G_T\}, \quad (2.47)$$

where

$$G_t = (V_t, E_t, X_t), \quad (2.48)$$

denotes the graph at time step  $t$ . Here  $V_t$  indicates the node set at time  $t$ ,  $E_t$  indicates the edge set at time  $t$ ,  $X_t \in \mathbb{R}^{N_t \times d_0}$  indicates the node feature matrix at time  $t$ . The number of nodes  $N_t$  may vary across time steps.

At each time step  $t$ , a shared GNN encoder is applied to  $G_t$ . It is a 2-layer GAT regression model. The first GAT layer is

$$z_{t,i}^{(1)} = W^{(1)}h_{t,i}^{(0)} + b^{(1)}, \quad (2.49)$$

$$e_{t,ij}^{(1)} = \text{LeakyReLU} \left( a^{(1)\top} [z_{t,i}^{(1)} \parallel z_{t,j}^{(1)}] \right), \quad j \in \mathcal{N}_t(i), \quad (2.50)$$

$$\alpha_{t,ij}^{(1)} = \frac{\exp(e_{t,ij}^{(1)})}{\sum_{m \in \mathcal{N}_t(i)} \exp(e_{t,im}^{(1)})}, \quad (2.51)$$

$$h_{t,i}^{(1)} = \text{ELU} \left( \sum_{j \in \mathcal{N}_t(i)} \alpha_{t,ij}^{(1)} z_{t,j}^{(1)} \right). \quad (2.52)$$

And the second GAT layer is

$$z_{t,i}^{(2)} = W^{(2)} h_{t,i}^{(1)} + b^{(2)}, \quad (2.53)$$

$$e_{t,ij}^{(2)} = \text{LeakyReLU} \left( a^{(2)\top} [z_{t,i}^{(2)} \parallel z_{t,j}^{(2)}] \right), \quad j \in \mathcal{N}_t(i), \quad (2.54)$$

$$\alpha_{t,ij}^{(2)} = \frac{\exp(e_{t,ij}^{(2)})}{\sum_{m \in \mathcal{N}_t(i)} \exp(e_{t,im}^{(2)})}, \quad (2.55)$$

$$h_{t,i}^{(2)} = \text{ELU} \left( \sum_{j \in \mathcal{N}_t(i)} \alpha_{t,ij}^{(2)} z_{t,j}^{(2)} \right). \quad (2.56)$$

A graph-level representation is obtained via mean pooling as

$$g_t = \frac{1}{N_t} \sum_{i=1}^{N_t} h_{t,i}^{(2)}, \quad g_t \in \mathbb{R}^{d_g}. \quad (2.57)$$

Thus, the sequence of graph embeddings is

$$g_1, g_2, \dots, g_T. \quad (2.58)$$

The sequence  $\{g_t\}_{t=1}^T$  is fed into an LSTM. Let  $h_t^L \in \mathbb{R}^{d_h}$  indicates the hidden state, and  $c_t^L \in \mathbb{R}^{d_h}$  indicates the cell state. Then the forget gate is

$$f_t = \sigma(W_f g_t + U_f h_{t-1}^L + b_f). \quad (2.59)$$

The input gate is

$$i_t = \sigma(W_i g_t + U_i h_{t-1}^L + b_i). \quad (2.60)$$

Then we get the cell state as

$$\tilde{c}_t = \tanh(W_c g_t + U_c h_{t-1}^L + b_c), \quad (2.61)$$

$$c_t^L = f_t \odot c_{t-1}^L + i_t \odot \tilde{c}_t. \quad (2.62)$$

The output gate is

$$o_t = \sigma(W_o g_t + U_o h_{t-1}^L + b_o). \quad (2.63)$$

The hidden state is

$$h_t^L = o_t \odot \tanh(c_t^L). \quad (2.64)$$

for  $t = 1, \dots, T$ .

We use the final hidden state

$$r = h_T^L, \quad (2.65)$$

and compute the output

$$\hat{y} = W_{\text{out}} r + b_{\text{out}}. \quad (2.66)$$

Repeat the above steps for all graph samples to get a prediction set  $\{\hat{y}_1, \hat{y}_2, \hat{y}_3, \dots, \hat{y}_N\}$  corresponding to the target set  $\{y_1, y_2, y_3, \dots, y_N\}$  where  $N = 50$  indicates the number of samples in a batch (or the number of epoch if no batching is applied). To determine the MSE, same as [Equation 2.41](#), we have

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2. \quad (2.67)$$

Thus, the full model in [Figure 2.7](#) can be expressed in a compact way as

$$\hat{y} = F_{\text{out}}\left(F_{\text{LSTM}}(F_{\text{GNN}}(G_1), \dots, F_{\text{GNN}}(G_T))\right). \quad (2.68)$$

The configured dimensions are:

1. Node feature dimension:  $d_{\text{in}} = 31$ ,
2. Number of predicted parameters:  $d_{\text{out}} = 2$ ,
3. GNN hidden dimension:  $d_{\text{hidden}} = 2d_{\text{in}}$ ,
4. Latent graph embedding:  $z = 2d_{\text{in}}$ ,
5. LSTM hidden dimension:  $d_{\text{lstm}} = 2d_{\text{in}}$ .

Note that the transformations applied to the parameters, node features, and summary statistics during different stages for this strategy are defined the same as [Table 2.2](#), [Table 2.3](#), and [Table 2.4](#), respectively.

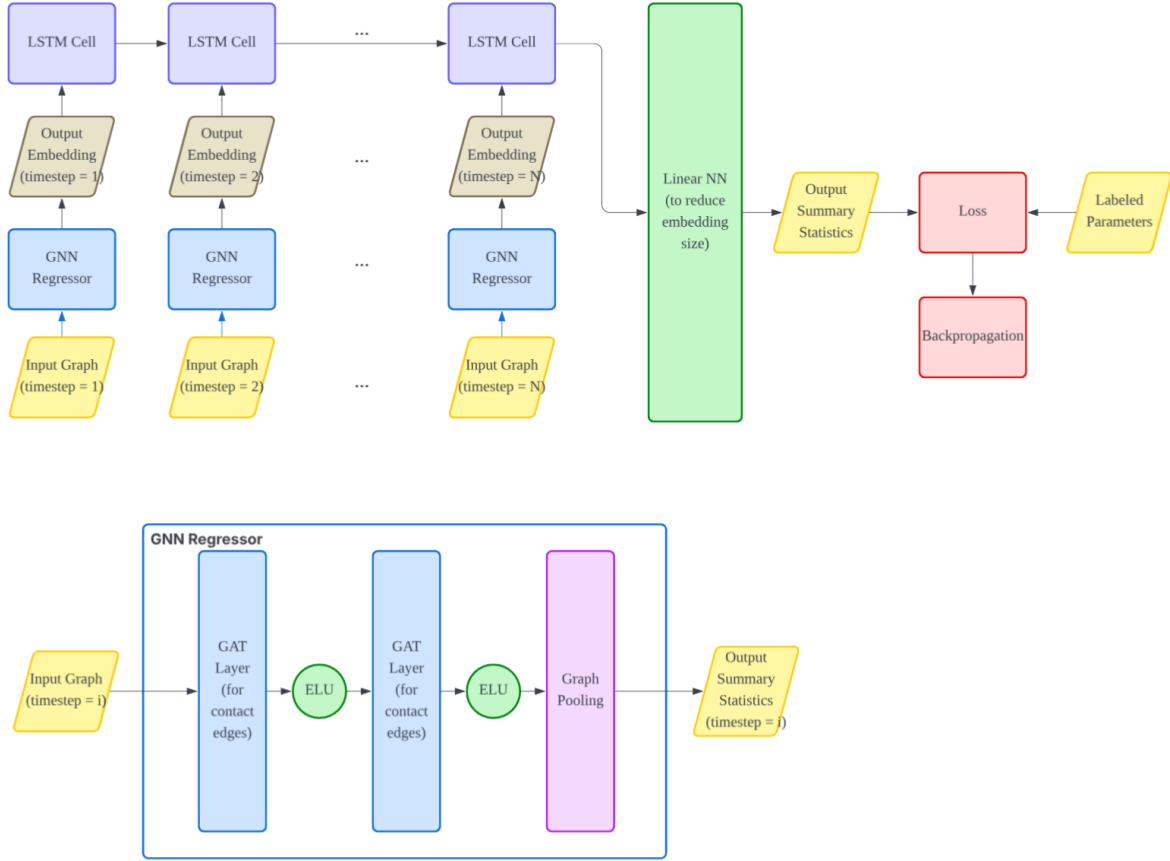


Figure 2.7: The architecture of the second strategy: GNN-LSTM regressor in the ABC. This architecture explains what's inside the trained GNN model [Figure 2.3](#), which is the pipeline overview flowchart in [section 2.3](#). The output summary statistics are the parameter gamma and alpha predicted by the GNN regressor. The labeled targets are the parameters gamma and alpha as well. The GNN regressor architecture is fixed over sequential graphs.

### 2.4.4 Third Strategy: Graph Neural Network Representation Learning in Approximate Bayesian Computation

The third strategy shares the same structural design as the GNN regressor in [subsection 2.4.2](#), but replaces the mean squared error (MSE) loss used for regression with a mixed triplet loss for representation learning. The model generates graph embeddings that serve as summary statistics within the overall ABC pipeline. The embedding dimension is fixed at 10 in this study to reduce computational cost, although theoretically larger embedding dimensions (closer to the number of input features) may yield improved training performance. The model in this strategy does not directly predict parameters. It learns a geometry of graph similarity and generates graph embeddings. Note that all the details of the following formulations can be referred to [subsection 1.3.5](#). To understand [Figure 2.9](#), we suppose that for any input graph  $G$ , the encoder produces a graph-level embedding  $z_G$ .

Then the input node features are in

$$H^{(0)} = \begin{bmatrix} h_1^{(0)} \\ h_2^{(0)} \\ \vdots \\ h_N^{(0)} \end{bmatrix}. \quad (2.69)$$

The GAT layers are the same as [subsection 2.4.2](#), so given the input of layer 1 as

$$z_i^{(1)}. \quad (2.70)$$

We can get the output of layer 4 as

$$h_i^{(4)} = GATs(z_i^{(1)}). \quad (2.71)$$

Then we get the graph pooling as

$$h_G = \frac{1}{N} \sum_{i=1}^N h_i^{(4)}, \quad (2.72)$$

and we reduce the graph embedding size to 6 by

$$z_G = W_{\text{proj}} h_G + b_{\text{proj}}. \quad (2.73)$$

Given a triplet of graphs as

$$G^{(a)}, \quad G^{(p)}, \quad G^{(n)}. \quad (2.74)$$

We compute embeddings

$$z^{(a)} = \phi(G^{(a)}), \quad z^{(p)} = \phi(G^{(p)}), \quad z^{(n)} = \phi(G^{(n)}). \quad (2.75)$$

The anchor-positive Euclidean distance is defined as

$$d_{ap} = \|z^{(a)} - z^{(p)}\|_2 = \sqrt{\sum_{k=1}^d \left(z_k^{(a)} - z_k^{(p)}\right)^2}. \quad (2.76)$$

The anchor-negative Euclidean distance is defined as

$$d_{an} = \|z^{(a)} - z^{(n)}\|_2 = \sqrt{\sum_{k=1}^d \left(z_k^{(a)} - z_k^{(n)}\right)^2}. \quad (2.77)$$

Then the Euclidean triplet loss is defined as

$$\ell_{\text{Euclidean}} = \max(0, d_{ap} - d_{an} + \text{margin}_1). \quad (2.78)$$

The anchor-positive similarity is defined as

$$\text{cos}_{ap} = \frac{\sum_{k=1}^d z_k^{(a)} z_k^{(p)}}{\sqrt{\sum_{k=1}^d (z_k^{(a)})^2} \cdot \sqrt{\sum_{k=1}^d (z_k^{(p)})^2}}. \quad (2.79)$$

The anchor-negative similarity is defined as

$$\text{cos}_{an} = \frac{\sum_{k=1}^d z_k^{(a)} z_k^{(n)}}{\sqrt{\sum_{k=1}^d (z_k^{(a)})^2} \cdot \sqrt{\sum_{k=1}^d (z_k^{(n)})^2}}. \quad (2.80)$$

Then the cosine triplet loss is defined as

$$\ell_{\text{cosine}} = \max(0, (1 - \text{cos}_{ap}) - (1 - \text{cos}_{an}) + \text{margin}_2). \quad (2.81)$$

Thus, the mixed triplet loss is defined as

$$\ell_{\text{mixed}} = \ell_{\text{Euclidean}} + \alpha \ell_{\text{cosine}}. \quad (2.82)$$

Repeat the above steps for all graph triplets to get a mixed triplet loss set  $\{ \ell_{\text{mixed}}^{(1)}, \ell_{\text{mixed}}^{(2)}, \ell_{\text{mixed}}^{(3)}, \dots, \ell_{\text{mixed}}^{(N)} \}$  where  $N = 50$  indicates the number of samples in a batch (or the number of epoch if no batching is applied). To determine the batch (or epoch loss if no batching is applied),

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N \ell_{\text{mixed}}^{(i)}. \quad (2.83)$$

Thus, the overall idea of training is

$$\min_{\theta} \mathcal{L}, \quad (2.84)$$

where  $\theta$  includes:

1. GAT weight matrices  $W_l^{(k)}, W_c^{(k)}$ .
2. Attention vectors  $a_l^{(k)}, a_c^{(k)}$ .
3. Bias terms  $b_l^{(k)}, b_c^{(k)}$ .
4. Projection parameters  $W_{\text{proj}}, b_{\text{proj}}$ .

The triplets are constructed by randomly sampling from the dataset. Given a dataset  $D$  containing labeled samples, the first triplet sampling method is

1. Randomly select two class indices  $i$  and  $l$  such that  $i \neq l$ .
2. Randomly select an anchor sample  $j$  from class  $D[i]$ .
3. Randomly select a positive sample  $k \neq j$  from class  $D[i]$ .
4. Randomly select a negative sample  $m$  from class  $D[l]$ .
5. Construct the triplet  $(X_{\text{anc}}, X_{\text{pos}}, X_{\text{neg}})$ .
6. Repeat until the desired number of triplets  $N$  is obtained.

This procedure ensures that the anchor and positive samples belong to the same class, while the negative sample is drawn from a different class [13]. Note that this approach is implemented by Sushma Dhamodharan, one of the previous master's students in our research group.

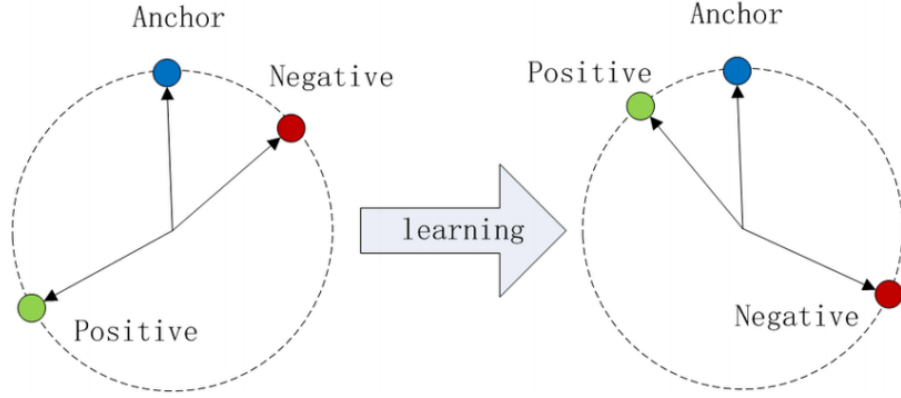


Figure 2.8: Visualization of triplet loss [36].

Let  $D = \{D_1, D_2, \dots, D_C\}$  denote the dataset, where each  $D_i$  contains samples generated from the same parameter set. Then the second triplet sampling method is

1. For each parameter group  $D_i$  in the dataset:
2. Enumerate all unordered pairs of samples  $(x_j, x_k)$  from  $D_i$  such that  $j \neq k$ . These pairs serve as anchor–positive candidates.
3. Assign the anchor and positive samples:

$$X_{\text{anc}} = x_j, \quad X_{\text{pos}} = x_k \quad (2.85)$$

4. Randomly select a different parameter group  $D_l$  such that  $l \neq i$ .
5. Randomly select a negative sample  $x_m$  from  $D_l$  and assign

$$X_{\text{neg}} = x_m \quad (2.86)$$

6. Construct the triplet

$$(X_{\text{anc}}, X_{\text{pos}}, X_{\text{neg}}) \quad (2.87)$$

7. Repeat this process for all anchor–positive pairs in  $D_i$ .

This procedure ensures that the anchor and positive samples originate from the same parameter group, while the negative sample is drawn from a different parameter group. By enumerating all within-group pairs, this method is less computationally heavy and systematically explores all anchor–positive relationships while maintaining stochasticity in the negative sampling.

The configured parameters:

1. Node feature dimension:  $d_{\text{in}} = 31$ ,
2. Hidden dimension:  $d_{\text{hidden}} = 4d_{\text{in}}$ ,
3. Embedding dimension:  $d_{\text{out}} = 6$ .
4. mixture weight:  $\alpha = 0.5$ ,
5. Euclidean margin:  $\text{margin}_1 = 1$ ,
6. cosine margin:  $\text{margin}_2 = 0.2$ .

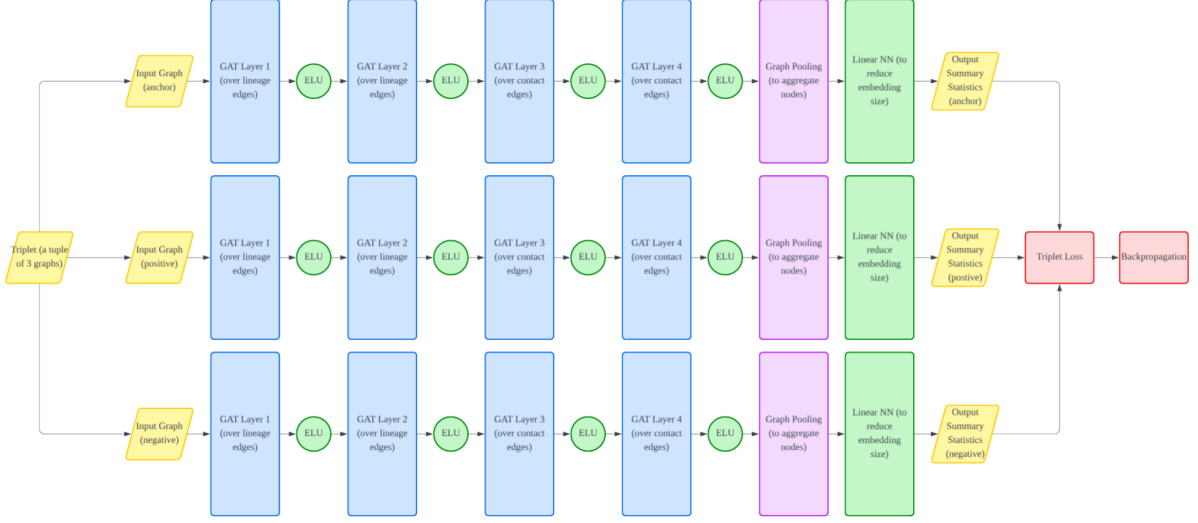


Figure 2.9: The architecture of the third strategy: GNN embedding learner in the ABC. This architecture explains what’s inside the trained GNN model in Figure 2.3, which is the pipeline overview flowchart in section 2.3. The output summary statistics are graph-level representations generated by the GNN embedding learner. The triplets are composed of anchor graphs, positive graphs and negative graphs.

| Stages              | $\theta$ transformation                         |
|---------------------|-------------------------------------------------|
| Beginning           | $\log_{10}(\theta)$                             |
| <b>CellModeller</b> | $\theta$                                        |
| GNN begins          | $\log_{10}(\theta)$                             |
| GNN ends            | $g = \text{READOUT}(\{h_i \mid i \in V\})$      |
| ABC begins          | $g' = \frac{g - g_{\min}}{g_{\max} - g_{\min}}$ |
| ABC ends            | $\log_{10}(\theta)$                             |

Table 2.5: Transformations applied to the parameter  $\theta$  during different stages for the embedding learning tasks (i.e., the third and the fourth strategy) with relative distance of summary statistics. The  $g$  stands for the graph embedding, which aggregates all the node embeddings across the graph. The  $h_i \in \mathbb{R}^d$  denotes the node embedding, and the  $g \in \mathbb{R}^d$  denotes the graph embedding. Aggregation could be the mean graph pooling, which means  $g = \frac{1}{|V|} \sum_{i \in V} h_i$ . Note that the summary statistics require normalization, even when the relative distance is used. There is no standard score normalization for the node targets here because the triplets are applied as graph labels.

| Stages     | Node features $X$ | transformation               |
|------------|-------------------|------------------------------|
| GNN begins |                   | $Z = \frac{X - \mu}{\sigma}$ |
| GNN ends   |                   | $X$                          |

Table 2.6: Transformations applied to the node features  $X$  during different stages for the embedding learning tasks (i.e., the third and the fourth strategy). Note that  $X$  means the node feature matrix that includes each node feature  $x$ . This table remains the same as in the first and second strategies.

| Stages     | Summary statistics | $S_\phi(x)$ transformation                                                            |
|------------|--------------------|---------------------------------------------------------------------------------------|
| ABC begins |                    | $\tilde{S}_\phi(x) = \frac{S_\phi(x) - S_\phi^{\min}}{S_\phi^{\max} - S_\phi^{\min}}$ |
| ABC ends   |                    | $\tilde{S}_\phi(x) = \frac{S_\phi(x) - S_\phi^{\min}}{S_\phi^{\max} - S_\phi^{\min}}$ |

Table 2.7: Transformations applied to the summary statistics  $S_\phi(x)$  during different stages for the embedding learning tasks (i.e., the third and the fourth strategy). The summary statistics here are graph representations aggregated by the graph pooling mechanism over all nodes in the graph. Note that the summary statistics require normalization, even when the relative distance is used.

### 2.4.5 Fourth Strategy: Graph Neural Network Representation Learning in Approximate Bayesian Computation with Long Short-term Memory

The fourth strategy is also a graph-level representation-learning task, in which the model generates a graph embedding for each input graph, similar to [subsection 2.4.4](#). Instead of using lineage edges, it applies LSTM layers to capture temporal information in the growing population dynamics, similar to [subsection 2.4.3](#). Note that all the details of the following formulations can be referred to [subsection 1.3.2](#) and [subsection 1.3.5](#). To understand [Figure 2.10](#), we consider a sequence of graph triplets

$$\left(G_1^{(a)}, G_1^{(p)}, G_1^{(n)}\right), \left(G_2^{(a)}, G_2^{(p)}, G_2^{(n)}\right), \dots, \left(G_T^{(a)}, G_T^{(p)}, G_T^{(n)}\right), \quad (2.88)$$

where  $G_t^{(a)}$  indicates the anchor graph at time step  $t$ ;  $G_t^{(p)}$  indicates the positive graph at time step  $t$ ;  $G_t^{(n)}$  indicates the negative graph at time step  $t$ . So, each branch forms a graph sequence.

Then at time step  $t$ , the GAT layers are the same as [subsection 2.4.3](#), so given the input of layer 1 as

$$z_i^{(1)}. \quad (2.89)$$

We can get the output of layer 2 as

$$h_i^{(2)} = GATs(z_i^{(1)}). \quad (2.90)$$

Then same as [subsection 2.4.3](#), we have graph pooling as

$$g_t = \frac{1}{N_t} \sum_{i=1}^{N_t} h_{t,i}^{(2)}. \quad (2.91)$$

Then we have the graph embedding sequences with respect to the anchor, positive, and negative graph samples as

$$g_t^{(a)} = \phi(G_t^{(a)}), \quad g_t^{(p)} = \phi(G_t^{(p)}), \quad g_t^{(n)} = \phi(G_t^{(n)}). \quad (2.92)$$

$$\{g_t^{(a)}\}_{t=1}^T, \quad \{g_t^{(p)}\}_{t=1}^T, \quad \{g_t^{(n)}\}_{t=1}^T. \quad (2.93)$$

LSTMs are applied in the same way as [subsection 2.4.3](#), but instead of one LSTM branch, we have three LSTM branches as follows. The anchor branch is defined as the LSTMs with respect to the anchor graph sequences, so

$$f_t^{(a)} = \sigma(W_f^{(a)} g_t^{(a)} + U_f^{(a)} h_{t-1}^{L,a} + b_f^{(a)}), \quad (2.94)$$

$$i_t^{(a)} = \sigma(W_i^{(a)} g_t^{(a)} + U_i^{(a)} h_{t-1}^{L,a} + b_i^{(a)}), \quad (2.95)$$

$$\tilde{c}_t^{(a)} = \tanh(W_c^{(a)} g_t^{(a)} + U_c^{(a)} h_{t-1}^{L,a} + b_c^{(a)}), \quad (2.96)$$

$$c_t^{L,a} = f_t^{(a)} \odot c_{t-1}^{L,a} + i_t^{(a)} \odot \tilde{c}_t^{(a)}, \quad (2.97)$$

$$o_t^{(a)} = \sigma(W_o^{(a)} g_t^{(a)} + U_o^{(a)} h_{t-1}^{L,a} + b_o^{(a)}), \quad (2.98)$$

$$h_t^{L,a} = o_t^{(a)} \odot \tanh(c_t^{L,a}), \quad (2.99)$$

$$r^{(a)} = h_T^{L,a}. \quad (2.100)$$

The positive and negative branches are defined as the LSTMs with respect to the positive and negative graph sequences in the same way, yielding

$$r^{(p)} = h_T^{L,p}, \quad r^{(n)} = h_T^{L,n}. \quad (2.101)$$

Then the linear neural network layer is applied to them to reduce the embedding dimension to 6 which can be customized. That gives

$$z^{(a)} = W_{\text{proj}} r^{(a)} + b_{\text{proj}}, \quad (2.102)$$

$$z^{(p)} = W_{\text{proj}} r^{(p)} + b_{\text{proj}}, \quad (2.103)$$

$$z^{(n)} = W_{\text{proj}} r^{(n)} + b_{\text{proj}}. \quad (2.104)$$

Then apply the same triplet loss function [Equation 2.82](#) that is defined in [subsection 2.4.4](#), so we have

$$\ell_{\text{mixed}}, \quad (2.105)$$

for each graph triplet. And we repeat the above steps for all graph triplets to get a mixed triplet loss set  $\{ \ell_{\text{mixed}}^{(1)}, \ell_{\text{mixed}}^{(2)}, \ell_{\text{mixed}}^{(3)}, \dots, \ell_{\text{mixed}}^{(N)} \}$  where  $N = 50$  indicates the number of samples in a batch (or the number of epoch if no batching is applied). To determine the batch (or epoch loss if no batching is applied),

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^N \ell_{\text{mixed}}^{(i)}. \quad (2.106)$$

Thus, the overall idea of training is

$$\min_{\theta} \mathcal{L}, \quad (2.107)$$

where  $\theta$  includes all GNN, LSTM, attention, and projection parameters.

Note that the graph triplet sampling methods are the same as in [subsection 2.4.4](#).

The configured parameters:

1. Node feature dimension:  $d_{\text{in}} = 31$ ,
2. Embedding dimension:  $d = 6$ ,
3. GNN hidden dimension:  $d_{\text{hidden}} = 2d_{\text{in}}$ ,
4. Latent graph embedding dimension:  $z = 2d_{\text{in}}$ ,
5. LSTM hidden dimension:  $d_{\text{lstm}} = 2d_{\text{in}}$ .
6. mixture weight:  $\alpha = 0.5$ ,
7. Euclidean triplet margin:  $\text{margin}_1 = 1$ ,
8. cosine margin:  $\text{margin}_2 = 0.2$ .

Note that the transformations applied to the parameters, node features, and summary statistics during different stages for this strategy are defined the same as [Table 2.5](#), [Table 2.6](#), and [Table 2.7](#), respectively.

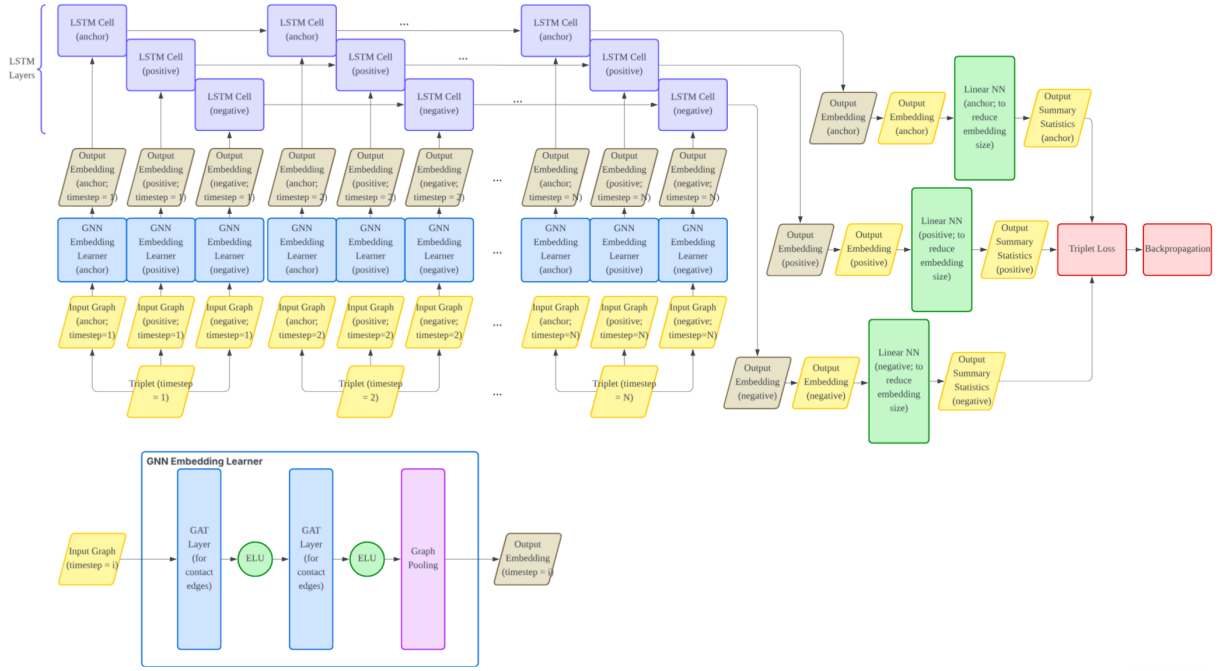


Figure 2.10: The architecture of the fourth strategy: GNN-LSTM embedding learner in the ABC. This architecture explains what’s inside the trained GNN model, as shown in Figure 2.3, which is the pipeline overview flowchart in section 2.3. The output summary statistics are graph-level representations generated by the GNN embedding learner. The triplets are composed of anchor graphs, positive graphs, and negative graphs. The GNN embedding learner architecture is fixed over sequential graphs.

## 2.4.6 Fifth Strategy: Conventional Approximate Bayesian Computation Method

In the fifth strategy (S5), six traditional summary statistics are used in the ABC pipeline based on the results of the sensitivity analysis of previous research [2]. These statistics are manually selected based on expert biological knowledge and require substantial domain expertise and effort.

This approach does not require any training tasks because the ABC uses only traditional summary statistics, which are manually selected and determined by humans based on biological meaning and context in this project. This approach requires fewer implementation steps and runs much faster. Traditional summary statistics should be selected based

on their informativeness, dimensionality, and interpretability. It is a common challenge for biologists to develop informative summary statistics. The advantages of this strategy are its dimensionality and interpretability.

| Stages              | $\theta$ transformation                                                 |
|---------------------|-------------------------------------------------------------------------|
| Beginning           | $x = \log_{10}(\theta)$ .                                               |
| <b>CellModeller</b> | $\theta$                                                                |
| ABC begins          | $\tilde{x} = \frac{\log_{10}(\theta) - x_{\min}}{x_{\max} - x_{\min}},$ |
| ABC ends            | $\log_{10}(\theta)$                                                     |

Table 2.8: Transformations applied to the parameter  $\theta$  during different stages for the traditional approach (i.e., the fifth strategy) with relative distance of summary statistics. Note that  $x_{\min} = \min(\log_{10}(\theta))$  and  $x_{\max} = \max(\log_{10}(\theta))$ .

| Stages     | Summary statistics $S_\phi(x)$ transformation                |
|------------|--------------------------------------------------------------|
| ABC begins | $\tilde{S}(x) = \frac{S(x) - S_{\min}}{S_{\max} - S_{\min}}$ |
| ABC ends   | $\tilde{S}(x) = \frac{S(x) - S_{\min}}{S_{\max} - S_{\min}}$ |

Table 2.9: Transformations applied to the summary statistics  $S(x)$  during different stages for the traditional approach (i.e., the fifth strategy).

## 2.4.7 Summary

The training configurations for the first four strategies are identical in their shared components, except for the elements that distinguish each approach. Their training results cannot be directly compared because different loss criteria are used (absolute versus relative loss functions).

The first two strategies perform graph-level regression tasks to generate summary statistics that are subsequently used in their respective ABC pipelines. The regression outputs can be regarded as deterministic point estimates of the model parameters. However, the goal of this study extends beyond point estimation to include interval estimation and quantification of uncertainty. Therefore, the regression outputs (representing the point estimates of the parameters) are used as summary statistics within the overall ABC framework.

Strategies three and four employ graph-level representation learning tasks to generate graph embeddings as summary statistics for the ABC pipeline. These strategies use mixed

triplet loss as the training criterion, whereas the first two strategies use mean squared error (MSE). In other words, Strategies 1 and 2 adopt absolute loss functions for graph-level regression, while Strategies 3 and 4 adopt relative loss functions for graph-level representation learning.

Strategies 1 and 3 incorporate an additional edge type (i.e., the lineage edge type) to encode temporal information, thereby transforming time-series graph data into a single static graph and reducing computational cost. In contrast, Strategies 2 and 4 introduce Long Short-Term Memory (LSTM) layers to handle temporal information directly, enabling the time-series graph data to be processed sequentially and linked through LSTM memory mechanisms to capture temporal dependencies.

## 2.5 Posterior Comparisons over Strategies

### 2.5.1 Delta Function and Kernel Density Estimation

As described in [subsection 2.5.3](#), when the simulated-experimental data (described in [section 2.2](#)) are used to evaluate the performance of the inference pipeline, it is assumed that all gamma and alpha parameter values are fixed across all simulated-experimental samples. This means that the simulated-experimental samples should have the same gamma and alpha parameter values, although they are not identical due to the stochasticity of the simulations.

For the shape, mean, and variance of their parameter distributions, it is assumed that their gamma and alpha parameters follow a Dirac delta function [\[14\]](#).

The Dirac delta function, denoted by  $\delta(x)$ , is defined by two main properties. It is assumed that

$$\delta(x) = \begin{cases} \infty, & x = 0 \\ 0, & x \neq 0 \end{cases}. \quad (2.108)$$

And the integral is

$$\int_{-\infty}^{\infty} \delta(x) dx = 1. \quad (2.109)$$

It means that the delta function is zero everywhere except at  $x = 0$ , where it is infinitely concentrated such that the total area equals 1.

The KL divergence (described in [Equation 1.238](#)) is applied to compare the shape of the distributions, and the Mean Squared Distance (MSD) is applied to compare the mean and variance of the distributions. The following sections will show these two comparison metrics in detail. When the KL divergence is computed, the sample points are used to construct the nonparametric distribution via KDE (described in [Equation 1.233](#)) [\[46\]](#).

The posterior distribution obtained from the ABC-SMC algorithm is represented by a set of weighted samples

$$\{(\theta_i, w_i)\}_{i=1}^N, \quad (2.110)$$

where  $\theta_i$  denotes the sampled parameter values and  $w_i$  are normalized importance weights.

Since these samples provide only a discrete representation of the posterior distribution, a continuous density approximation is required to compute quantities such as the KL divergence.

To obtain a smooth estimate of the posterior density, we applied KDE (described in [Equation 1.233](#)). The KDE approximation of the posterior is given by

$$\hat{p}(\theta) = \sum_{i=1}^N w_i K_h(\theta - \theta_i), \quad (2.111)$$

where  $K_h(\cdot)$  is a kernel function with bandwidth parameter  $h$ . In this work, a Gaussian kernel is used:

$$K_h(\theta - \theta_i) = \frac{1}{(2\pi h^2)^{d/2}} \exp\left(-\frac{\|\theta - \theta_i\|^2}{2h^2}\right), \quad (2.112)$$

where  $d$  denotes the dimension of the parameter vector. This procedure produces a smooth approximation of the posterior distribution by representing it as a weighted mixture of Gaussian kernels centered at the sampled parameter values.

## 2.5.2 Gaussian Approximation of a Dirac Delta

As described in [subsection 2.5.3](#), to quantify how well the inferred posterior distributions concentrate around the true parameter values, we compute the KL divergence between the posterior distribution obtained from ABC-SMC and a reference distribution centered on the ground-truth parameter [\[35\]](#).

However, if we directly plug the Dirac delta function into the KL divergence, we will get a problematic expression.

Let  $\theta^*$  be the true parameter,  $x$  be the simulated data,  $S(x)$  be the summary statistics of simulated data  $x$ ,  $\delta_{\theta^*}(\theta)$  be the true posterior,  $p(\theta | S(x))$  be the ABC posterior, and  $p(\theta)$  be the prior. Then we get the problematic KL divergence as

$$D_{\text{KL}}(p(\theta | S(x)) \| \delta_{\theta^*}) = \int p(\theta | S(x)) \log \frac{p(\theta | S(x))}{\delta_{\theta^*}(\theta)} d\theta. \quad (2.113)$$

This is not well-defined (infinite) unless  $p(\theta | S(x))$  is itself a delta at  $\theta^*$ , because  $\delta_{\theta^*}(\theta) = 0$  for  $\theta \neq \theta^*$ . So the log term blows up

$$\log \frac{p(\theta | S(x))}{0} = +\infty. \quad (2.114)$$

Hence,

$$D_{\text{KL}}(p(\theta | S(x)) \| \delta_{\theta^*}) = +\infty. \quad (2.115)$$

To solve this problematic expression, we approximate the Dirac delta function with a narrow Gaussian distribution. Mathematically,

$$q_\varepsilon(\theta) = \mathcal{N}(\theta^*, \varepsilon^2 I), \quad (2.116)$$

where  $\varepsilon$  is a small positive constant that controls the width of the approximation and  $I$  is the identity matrix. As  $\varepsilon \rightarrow 0$ , the Gaussian distribution approaches the Dirac delta  $\delta(\theta - \theta^*)$  in the distributional sense.

The posterior distribution  $p(\theta | S(x))$  is approximated from the weighted ABC samples using KDE. Given weighted samples  $\{\theta_i, w_i, x_i\}_{i=1}^N$ , the KL divergence from the posterior to the reference distribution is approximated as

$$D_{\text{KL}}(p(\theta | S(x)) \| q_\varepsilon(\theta)) = \int p(\theta | S(x)) \log \frac{p(\theta | S(x))}{q_\varepsilon(\theta)} d\theta = \sum_{i=1}^N w_i \log \frac{p(\theta_i | S(x_i))}{q_\varepsilon(\theta_i)}. \quad (2.117)$$

In addition to the joint KL divergence over the full parameter vector, we also computed marginal KL divergences for each parameter dimension. This provides insight into how accurately each parameter is inferred relative to the reference value.

To ensure numerical stability, a lower bound is imposed when evaluating the density ratio to avoid division by zero. This procedure provides a quantitative measure of how tightly the posterior distribution concentrates around the true parameter values while avoiding the singular behavior associated with an exact Dirac delta distribution.

The KL divergence is a widely used measure of similarity between two probability distributions, whether parametric or nonparametric. The posterior results of the five strategies are compared with the delta function defined by the fixed parameters used in the simulated experimental data. The marginal KL divergence values indicate the similarity for the parameters  $\gamma$  and  $\alpha$  individually, while the joint KL divergence quantifies the general similarity for both parameters. However, the joint KL divergence is not equal to the sum of the marginal values unless the two parameters are statistically independent.

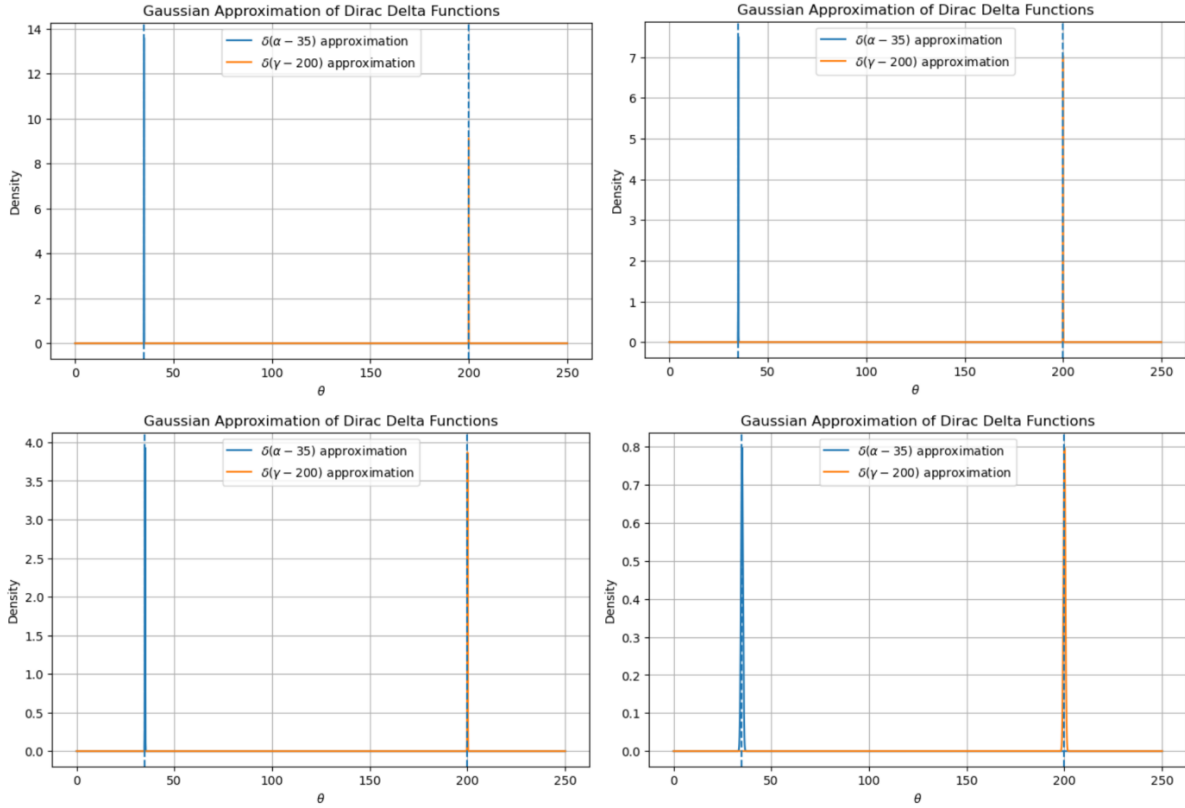


Figure 2.11: Gaussian approximation of Dirac delta functions. The orders are  $\varepsilon = 0.02$  (top-left),  $\varepsilon = 0.05$  (top-right),  $\varepsilon = 0.1$  (bottom-left), and  $\varepsilon = 0.1$  (bottom-right).

### 2.5.3 Simulation-based Validation in Approximate Bayesian Computation

There are many ways to check the informativeness of summary statistics. In this project, we applied simulation-based validation in an intuitive manner. Simulation-based calibration is another approach to assessing the informativeness of summary statistics, but we are not interested in it here. The following procedure is used.

1. Sample ground-truth parameters  $\theta^* \sim \pi(\theta)$ . If only one ground-truth parameter is selected to be the ground-truth baseline, then we assume there is no unknown noise.
2. Simulate synthetic data  $y^* \sim p(y^* | \theta^*)$  that is the same as  $p(x | \theta)$ . That means that we assume no model misspecification.
3. Compute summary statistics  $S(y^*)$ .
4. Pretend we do not know  $\theta^*$  and perform ABC inference using  $S(x)$  and  $S(y^*)$  to infer  $p(\theta | S(y^*))$
5. Evaluate whether the inferred posterior distribution recovers  $\theta^*$ .

This ground-truth parameter  $\theta^*$  promises that  $p(\theta | y^*)$  is a Dirac delta function with no variance, so  $Var(\theta | y^*) = 0$  here.

If  $S$  preserves information about  $\theta$ ,

$$p(\theta | S(D^*)) \approx p(\theta | D^*). \quad (2.118)$$

So, the posterior will peak near  $\theta^*$  since both  $x$  and  $y^*$  are generated from the same simulation module setups of **CellModeller**.

That means  $p(x | \theta)$  and  $p(y^* | \theta)$  are the same population  $p(D | \theta)$  where  $D$  indicates any arbitrary generated data given parameters in **CellModeller**.

That means

$$x, y^* \sim p(D | \theta). \quad (2.119)$$

And the posterior will have small variance  $Var(\theta | S(y^*)) \approx 0$  and we know  $Var(\theta | y^*) = 0$  so

$$Var(\theta | S(y^*)) \approx Var(\theta | y^*) = 0, \quad (2.120)$$

or

$$\Delta = \text{Var}(\theta \mid S(y^*)) - \text{Var}(\theta) \approx \text{Var}(\theta). \quad (2.121)$$

If  $S$  loses information,

$$p(\theta \mid S(y^*)) \approx p(\theta). \quad (2.122)$$

Then the posterior  $p(\theta \mid S(y^*))$  is wide, and it may not be centered at  $\theta^*$ . The posterior variance reduction  $\Delta \approx 0$  is very low because

$$\text{Var}(\theta \mid S(y^*)) \approx \text{Var}(\theta) = \frac{(b - a)^2}{12}, \quad (2.123)$$

where  $a$  is the upper bound of the parameter range, and  $b$  is the lower bound of the parameter range. It is assumed that a uniform distribution is selected as the prior  $\pi(\theta)$ .

#### 2.5.4 Effect of Unknown Noise and Model Mismatch

In reality, we replace the ground-truth baseline described in [subsection 2.5.3](#) with real observations in experiments. We then assume the presence of unknown noise and model mismatch, which could affect the inference results.

In practice, even if we assume there is no model mismatch, the observed data are often noisy

$$y_{\text{obs}} = y_{\text{true}} + \varepsilon, \quad (2.124)$$

where  $Y_{\text{true}} \sim p(y \mid \theta)$ , and  $\varepsilon \sim p_{\text{noise}}(\varepsilon)$ .

Mutual information becomes

$$I(\theta; S(y_{\text{obs}})). \quad (2.125)$$

Since noise reduces information

$$I(\theta; S(y_{\text{obs}})) \leq I(\theta; S(y_{\text{true}})). \quad (2.126)$$

If noise dominates

$$I(\theta; S(y_{\text{obs}})) \approx 0. \quad (2.127)$$

Then inference becomes impossible.

Note that simulations also introduce noise, but it is known and easy to control. A large noise in simulations can also cause issues, such as data  $x$  is no longer related to

the parameter  $\theta$  because noise dominates the patterns. In fact, both known noise from simulations and unknown noise from observations can dominate the information in the data, so ABC would fail because it cannot find useful patterns in such noisy data.

In practice, even if we assume that there is no unknown noise in the observations, model mismatch remains a common issue.

Assume the true data-generating process is

$$y_{\text{obs}} \sim p_{\text{true}}(y \mid \theta). \quad (2.128)$$

And the simulator model (used in ABC) is

$$y \sim p_{\text{sim}}(y \mid \theta). \quad (2.129)$$

Then the mismatch condition is

$$p_{\text{sim}}(y \mid \theta) \neq p_{\text{true}}(y \mid \theta). \quad (2.130)$$

As the acceptance threshold  $\varepsilon \rightarrow 0$ , ABC converges to:

$$\theta^* = \arg \min_{\theta} d(p_{\text{sim}}(\cdot \mid \theta), p_{\text{true}}(\cdot \mid \theta_{\text{true}})). \quad (2.131)$$

This is the best-fitting parameter under the simulator model, but it is generally not the true parameter

ABC inference targets incorrect parameters. Even if the mutual information is large, the information is only meaningful for the incorrect model.

$$I(\theta; S(y)) \text{ is large but only for } y \sim p_{\text{sim}}(y \mid \theta). \quad (2.132)$$

Noise can reduce the information in the data themselves  $x$ , and mismatch can destroy the correctness of inference, but they do not affect the informativeness of summary statistics. That means a good result in simulation-based validation described in [subsection 2.5.3](#) with a poor result on real observations of experiments indicates that the summary statistics are informative enough, but the poor results may be due to unknown noise or model mismatch.

## 2.5.5 Informativeness Measure by Simulation-based Validation Results

### General Case

There are many different ways to measure informativeness, and all of their details are described in [section 1.8](#).

In Semi-automatic ABC, the trained model's performance directly indicates the informativeness of the learned summary statistics. As described in [section 1.7](#), the optimal summary statistics are defined as  $S(x) = \mathbb{E}(\theta \mid x)$ , that is, the mean of the posterior  $p(\theta \mid x)$ .

However, the trained model performance can only be used for ML-based ABC, and it cannot be used to compare the informativeness of summary statistics between 2 different model architectures.

Thus, a straightforward metric we proposed is to measure informativeness as the difference between the ABC posterior and a ground-truth parameter, which serves as the baseline.

Given

$$I(\theta; S(x)) = \mathbb{E}_S [\text{KL}(p(\theta \mid S(x)) \parallel p(\theta))], \quad (2.133)$$

as described in [Equation 1.253](#).

We want to determine the relationship  $R$  between mutual information and simulation-based validation posterior results. That being said, we want to determine a relationship function  $R$  such that

$$I(\theta; S) = R(p(\theta \mid S(x)), q_\varepsilon(\theta)). \quad (2.134)$$

And show that the simulation-based validation posterior result  $(p(\theta \mid S(x)))$  and the ground-truth baseline  $q_\varepsilon(\theta)$  can directly tell the informativeness of summary statistics  $S$  by the mutual information  $I(\theta; S)$ .

In terms of the simulation-based validation posterior result  $(p(\theta \mid S(x)))$  and the ground-truth baseline  $q_\varepsilon(\theta)$ , we firstly consider the reverse KL divergence  $D_{\text{KL}}(q_\varepsilon(\theta), \parallel p(\theta \mid S(x)))$ . Mathematically,

$$D_{\text{KL}}(q_\varepsilon(\theta) \parallel p(\theta \mid S(x))) = \int q_\varepsilon(\theta) \log \frac{q_\varepsilon(\theta)}{p(\theta \mid S(x))} d\theta, \quad (2.135)$$

$$D_{\text{KL}}(q_\varepsilon \parallel p(\theta \mid S(x))) = \int q_\varepsilon(\theta) [\log q_\varepsilon(\theta) - \log p(\theta \mid S(x))] d\theta, \quad (2.136)$$

$$D_{\text{KL}}(q_\varepsilon \parallel p(\theta \mid S(x))) = \int q_\varepsilon(\theta) \log q_\varepsilon(\theta) d\theta - \int q_\varepsilon(\theta) \log p(\theta \mid S(x)) d\theta. \quad (2.137)$$

Note that

$$H(q_\varepsilon) = - \int q_\varepsilon(\theta) \log q_\varepsilon(\theta) d\theta. \quad (2.138)$$

So,

$$\int q_\varepsilon(\theta) \log q_\varepsilon(\theta) d\theta = -H(q_\varepsilon). \quad (2.139)$$

Then we get

$$D_{\text{KL}}(q_\varepsilon \parallel p(\theta \mid S(x))) = -H(q_\varepsilon) - \int q_\varepsilon(\theta) \log p(\theta \mid S(x)) d\theta, \quad (2.140)$$

where  $H(q_\varepsilon)$  is the differential entropy of  $q_\varepsilon$ .

$$D_{\text{KL}}(q_\varepsilon \parallel p(\theta \mid S(x))) = -H(q_\varepsilon) - \mathbb{E}_{\theta \sim q_\varepsilon} [\log p(\theta \mid S(x))], \quad (2.141)$$

$$D_{\text{KL}}(q_\varepsilon \parallel p(\theta \mid S(x))) \approx -H(q_\varepsilon) - \log p(\theta^* \mid S(x)), \quad (2.142)$$

for small  $\varepsilon$ .

We know

$$I(\theta; S(x)) = \mathbb{E}_{\theta^* \sim p(\theta), x \sim p(x \mid \theta^*)} \left[ \log \frac{p(\theta^* \mid S(x))}{p(\theta^*)} \right]. \quad (2.143)$$

Note that

$$D_{\text{KL}}(q_\varepsilon \parallel p(\theta \mid S(x))) = -H(q_\varepsilon) - \int q_\varepsilon(\theta) \log p(\theta \mid S(x)) d\theta, \quad (2.144)$$

$$D_{\text{KL}}(q_\varepsilon \parallel p(\theta)) = -H(q_\varepsilon) - \int q_\varepsilon(\theta) \log p(\theta) d\theta. \quad (2.145)$$

So,

$$D_{\text{KL}}(q_\varepsilon \| p(\theta)) - D_{\text{KL}}(q_\varepsilon \| p(\theta | S(x))) = \int q_\varepsilon(\theta) \log \frac{p(\theta | S(x))}{p(\theta)} d\theta. \quad (2.146)$$

As  $\varepsilon \rightarrow 0$

$$\int q_\varepsilon(\theta) \log \frac{p(\theta | S(x))}{p(\theta)} d\theta \longrightarrow \log \frac{p(\theta^* | S(x))}{p(\theta^*)}. \quad (2.147)$$

So,

$$D_{\text{KL}}(q_\varepsilon \| p(\theta)) - D_{\text{KL}}(q_\varepsilon \| p(\theta | S(x))) = \log \frac{p(\theta^* | S(x))}{p(\theta^*)}. \quad (2.148)$$

Then,

$$\mathbb{E}_{\theta^*, x} [D_{\text{KL}}(q_\varepsilon \| p(\theta)) - D_{\text{KL}}(q_\varepsilon \| p(\theta | S(x)))] = \mathbb{E}_{\theta^*, x} \left[ \log \frac{p(\theta^* | S(x))}{p(\theta^*)} \right] \quad (2.149)$$

$$= \mathbb{E}_{\theta^*, x} [KL] \quad (2.150)$$

$$= I(\theta; S(x)). \quad (2.151)$$

Thus,

$$I(\theta; S(x)) \approx \mathbb{E}_{\theta^*, x} [D_{\text{KL}}(q_\varepsilon \| p(\theta)) - D_{\text{KL}}(q_\varepsilon \| p(\theta | S(x)))], \quad (2.152)$$

for small  $\varepsilon$ .

This implies that mutual information can be expressed as the expected decrease in KL divergence from a narrow  $q_\varepsilon$  to the posterior, relative to the prior.

## Uniform Prior

With a uniform prior on bounded support  $\Omega$ , and under the narrow-delta approximation,

$$I(\theta; S(x)) \approx \mathbb{E}_{\theta^*, x} [D_{\text{KL}}(q_\varepsilon \| \text{Unif}(\Omega)) - D_{\text{KL}}(q_\varepsilon \| p(\theta | S(x)))]. \quad (2.153)$$

Since the first term is constant, this is equivalent to

$$I(\theta; S(x)) = \text{constant} - \mathbb{E}_{\theta^* \sim p(\theta), x \sim p(x|\theta^*)} [D_{\text{KL}}(q_\varepsilon \| p(\theta | S(x)))]. \quad (2.154)$$

for small  $\varepsilon$ .

If we take  $N$  many ground-truth baseline  $\theta_i^*$  where  $i = 1, \dots, N$ , and simulate the associated data  $y_i$  then we have

$$\theta_i^* \sim p(\theta), \quad (2.155)$$

$$x_i \sim p(x \mid \theta_i^*). \quad (2.156)$$

If one sample  $(x_i, \theta_i^*)$  is taken as the observed data for 1 ABC run, then

$$\text{KL}_i = D_{\text{KL}}(q_\varepsilon^{(i)} \parallel p_i(\theta)), \quad (2.157)$$

where  $q_\varepsilon^{(i)}$  is centered at  $\theta_i^*$ .

$$\frac{1}{N} \sum_{i=1}^N \text{KL}_i \approx \mathbb{E}_{\theta^*, x} [D_{\text{KL}}(q_\varepsilon \parallel p(\theta \mid S(x)))], \quad (2.158)$$

for large  $N$ .

Thus,

$$I(\theta; S(x)) \approx \text{constant} - \frac{1}{N} \sum_{i=1}^N \text{KL}_i, \quad (2.159)$$

for small  $\varepsilon$  and large  $N$ .

### Approximation of Dirac Delta Function in Reverse KL Divergence

If we directly use the Dirac delta function, rather than using the approximation of it, then we have

$$\delta_{\theta^*}(\theta^*) = \infty. \quad (2.160)$$

So,

$$\log \delta_{\theta^*}(\theta^*) = +\infty. \quad (2.161)$$

Therefore,

$$D_{\text{KL}}(\delta_{\theta^*} \parallel p(\theta \mid S(x))) = +\infty. \quad (2.162)$$

That is why we still need  $q_\varepsilon(\theta)$  as the approximation of  $\delta_{\theta^*}$ , although the KL divergence is reversed here.

## Intuition of Non-reverse KL Divergence

In terms of the non-reverse KL divergence  $D_{\text{KL}}(p(\theta | S(x)) \| q_\varepsilon(\theta))$ , We define

$$\mathcal{L}_\varepsilon(S(x), \theta^*) := D_{\text{KL}}(p(\theta | S(x)) \| q_\varepsilon(\theta)), \quad (2.163)$$

as a score.

A smaller  $\mathcal{L}_\varepsilon$  means the posterior is closer to a narrow distribution centered at the true parameter  $\theta^*$ . If  $\mathcal{L}_\varepsilon$  is small, the posterior is both concentrated and centered near  $\theta^*$ .

Thus,  $\mathcal{L}_\varepsilon$  can be used as a truth-recovery loss, although it is not a direct measure of mutual information.

### 2.5.6 Mean Squared Error and Squared Error

In addition to the KL divergence, we evaluate the accuracy of the inferred posterior distribution using the mean squared distance to the ground-truth parameter value [6]. Let  $\theta$  denote the parameter vector and  $\theta_0$  the true parameter. The mean squared distance is defined as

$$\text{MSD} = \mathbb{E}_{p(\theta)} [\|\theta - \theta_0\|^2] = \int \|\theta - \theta_0\|^2 p(\theta) d\theta. \quad (2.164)$$

Note that MSD and MSE are the same. MSD consists of two components: a measure of variance and a measure of squared bias, in the context of interval estimation for the fixed parameters. It provides an alternative means of assessing similarity between two nonparametric distributions. The bias component represents the distance between the posterior mean and the fixed parameter values. This interval estimation (i.e., the posterior distribution) can therefore be interpreted as a combination of a point estimate (i.e., the posterior mean) and its associated uncertainty (i.e., variance).

In practice, the posterior distribution is represented by weighted samples  $\{(\theta_i, w_i)\}_{i=1}^N$  obtained from ABC-SMC, and the expectation is approximated as

$$\text{MSD} \approx \sum_{i=1}^N w_i \|\theta_i - \theta_0\|^2. \quad (2.165)$$

This quantity admits the bias–variance decomposition

$$\mathbb{E}[(\theta - \theta_0)^2] = \text{Var}(\theta) + (\mathbb{E}[\theta] - \theta_0)^2, \quad (2.166)$$

which shows that the mean squared distance reflects both the dispersion of the posterior (variance) and the systematic deviation of its mean from the true parameter value (bias).

If only the accuracy of the posterior mean is of interest, one may instead evaluate the squared error of the posterior mean,

$$\|\mathbb{E}_{p(\theta)}[\theta] - \theta_0\|^2, \quad (2.167)$$

where  $\theta_0$  denotes the true parameter value. Unlike the mean squared distance, this metric ignores the posterior variance and measures only how close the posterior center is to the ground-truth parameter. This is exactly the bias term in the MSE.

### 2.5.7 Replacement of Simulated-experimental Data with Real-experimental Data

When the simulated-experimental data are replaced with the real-experimental data, everything remains the same, except that we replace the approximated Dirac delta

$$q(\theta) = \mathcal{N}(\theta_0, \sigma^2 I), \quad (2.168)$$

with

$$\hat{q}(\theta) = \frac{1}{M} \sum_{j=1}^M K_h(\theta - \theta_j^{\text{exp}}), \quad (2.169)$$

such that

$$D_{\text{KL}}(p \parallel q) \approx \sum_{i=1}^N w_i \log \frac{\hat{p}(\theta_i)}{\hat{q}(\theta_i)}, \quad (2.170)$$

where  $w_i$  are the ABC-SMC posterior weights,  $\hat{p}$  is the KDE approximation of the posterior,  $\hat{q}$  is the KDE approximation of the experimental distribution. The weights remain the same because the expectation is still taken with respect to the posterior.

However, as mentioned earlier, real-experimental samples lack targets, so their associated gamma and alpha parameter values are unknown. Distribution comparison requires the targets of real-experimental samples, so the other evaluation approach is applied. A total of 24 parameter values are randomly sampled from the posterior and prior distributions, corresponding to the 24 available real experimental samples. The associated summary statistics are then generated and compared with those derived from the real experimental data. This evaluation approach is defined as sampling validation.

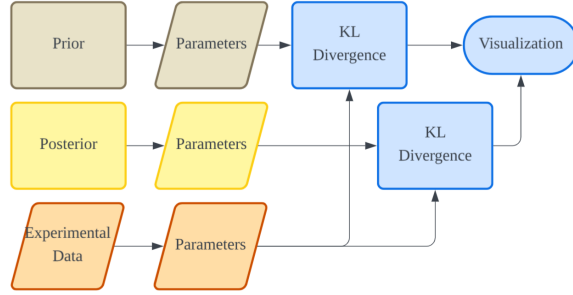


Figure 2.12: Sampling validation process flowchart for the first and second strategies. The summary statistics are the parameters themselves in the first and second strategies, so redundant and duplicate steps are removed to improve efficiency.

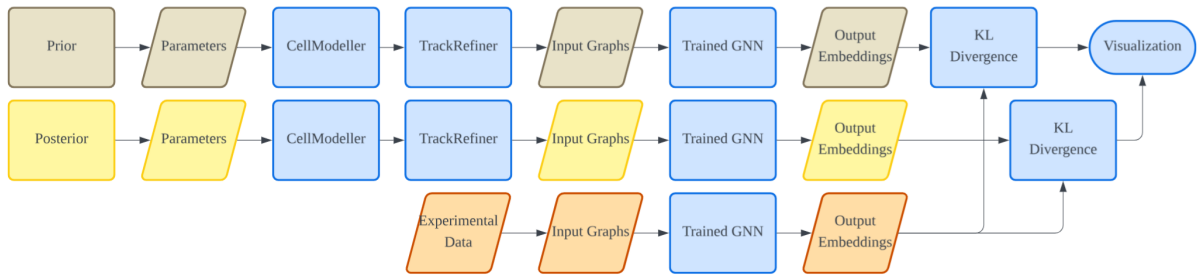


Figure 2.13: Sampling validation process flowchart for the third and fourth strategies. The summary statistics are the graph embeddings in the third and fourth strategies, so the trained GNNs are applied.

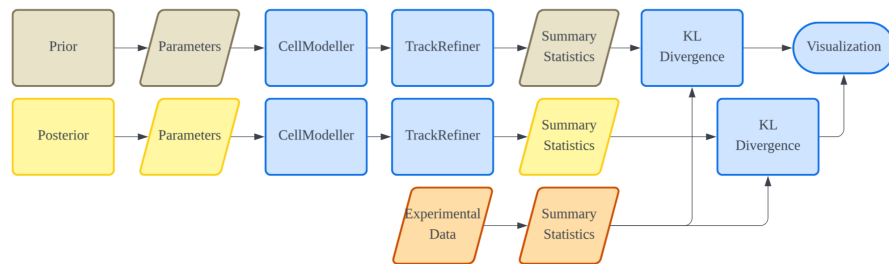


Figure 2.14: Sampling validation process flowchart for the fifth strategy. The summary statistics are the traditional ones in this strategy, so the trained GNNs are removed.

# Chapter 3

## Results

### 3.1 Feature Importance Ranking

There are 69 features as the output from `TrackRefiner` as described in [section 1.4](#). Only the top 57 features are involved in the feature importance ranking, since some constant and redundant features are removed beforehand as described in [section subsection 2.1.3](#). This result is obtained from the GAT-based attention feature importance as described in [subsection 2.1.4](#).

Below, we consider three separate importance ranks, based on different graph constructions:

1. both contact and lineage edge types;
2. contact edge type only;
3. lineage edge type only.

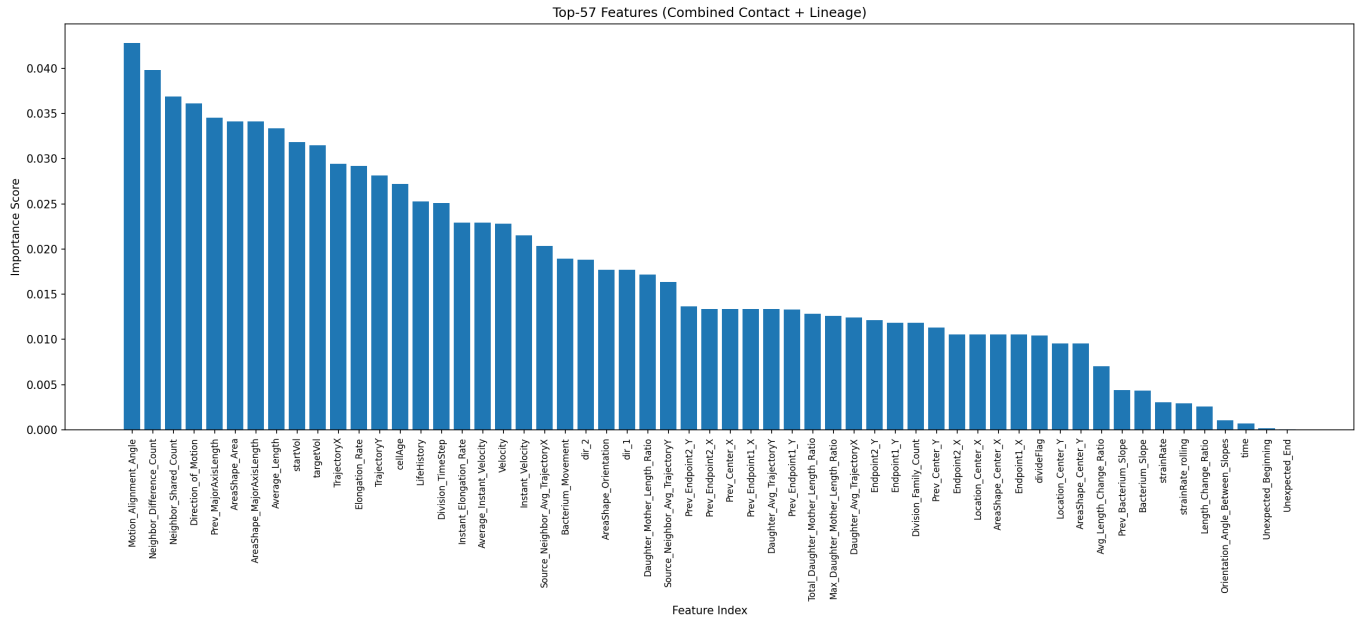


Figure 3.1: GAT-based attention feature importance rank in terms of both contact and lineage edge types.

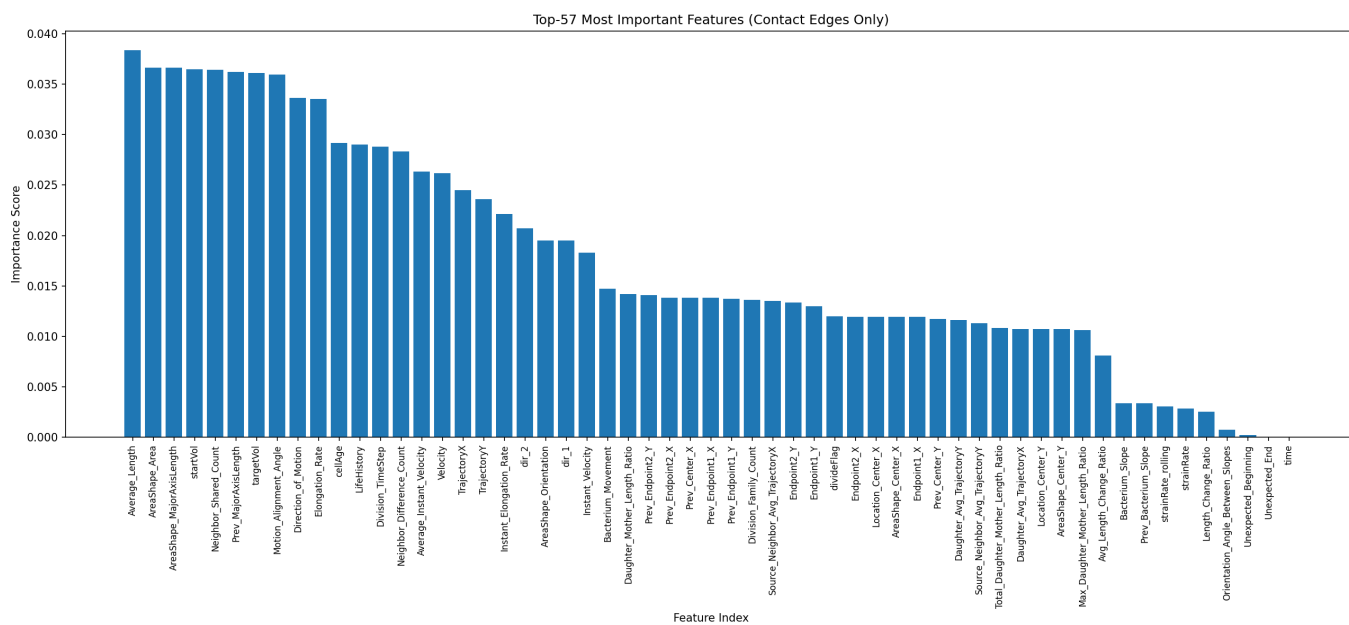


Figure 3.2: GAT-based attention feature importance rank in terms of the contact edge type only.

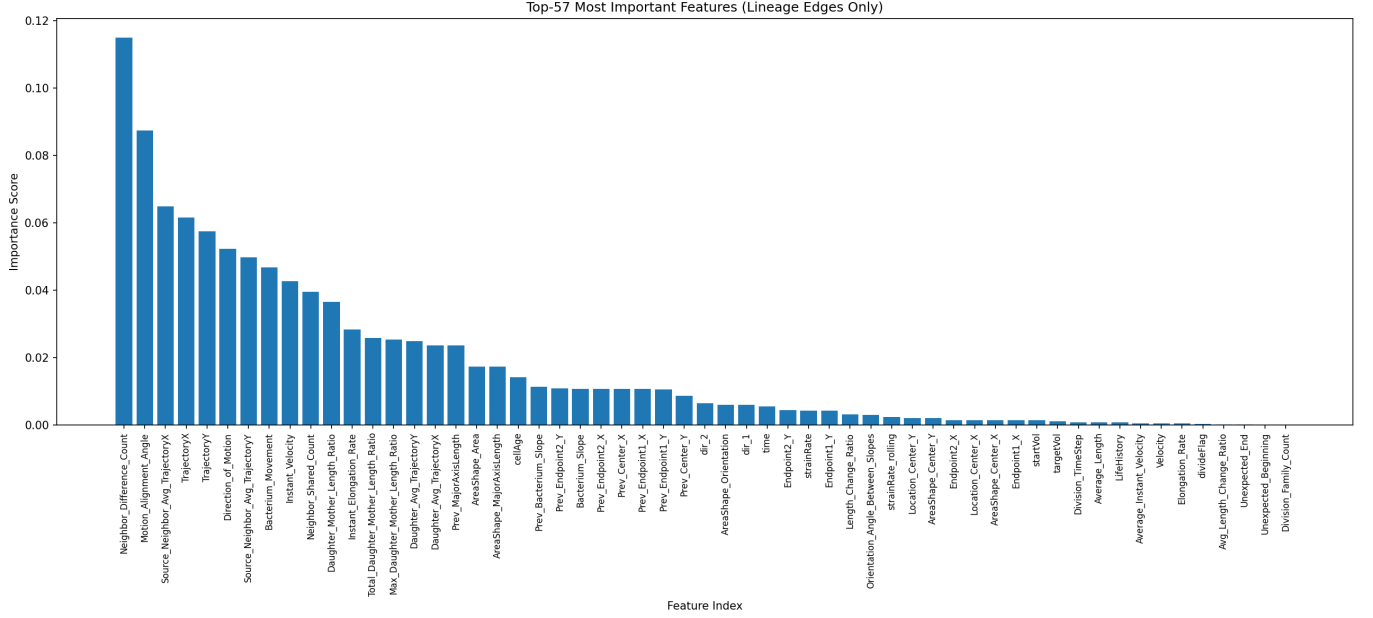


Figure 3.3: GAT-based attention feature importance rank in terms of the lineage edge type only.

## 3.2 Feature Selection

Based on the feature importance ranking result as described in [section 3.1](#), there are 57 features to be considered. It is time-consuming to fine-tune the number of features by considering all possible feature sets. In this case, only 6 feature sets are arbitrarily selected and applied: the most important 7, 17, 27, 37, 47, and 57. The training architecture of the first strategy is applied and kept fixed, as discussed in [subsection 2.4.2](#). There is no need to repeat this feature selection process for each strategy using different architectures, because this is to tell the relationship between features and targets, and the basic training architectures are very similar across all strategies. The dataset is also fixed across 6 feature sets.

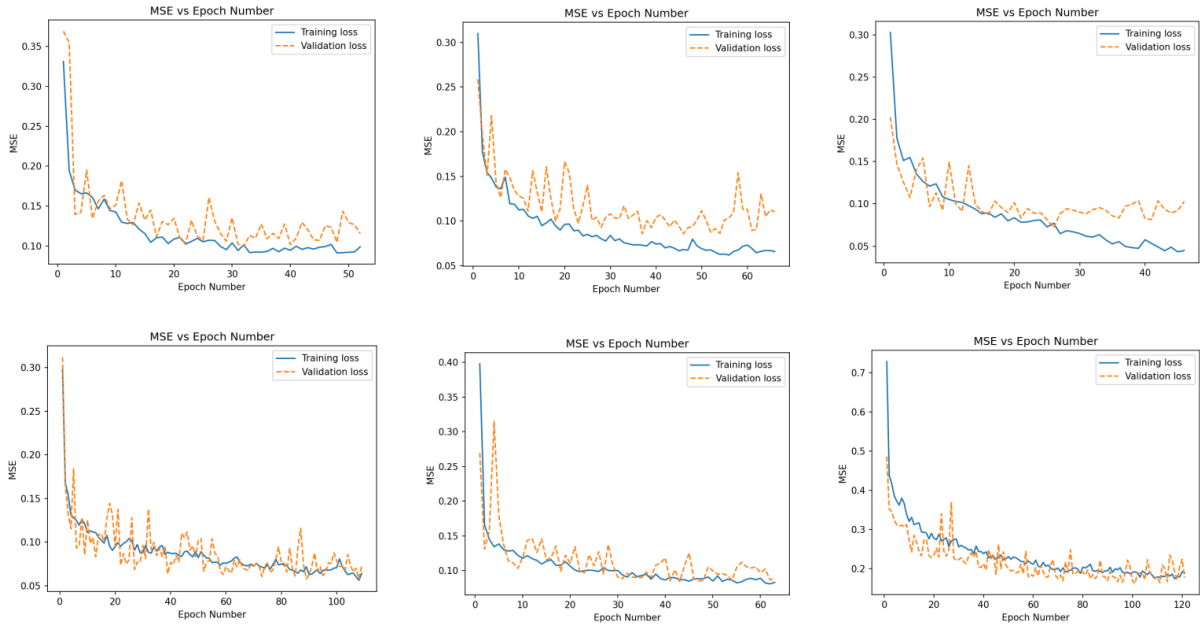


Figure 3.4: Training results of 6 cases. The orders are 57 features (top-left), 47 features (top-middle), 37 features (top-right), 27 features (bottom-left), 17 features (bottom-middle), and 7 features (bottom-right).

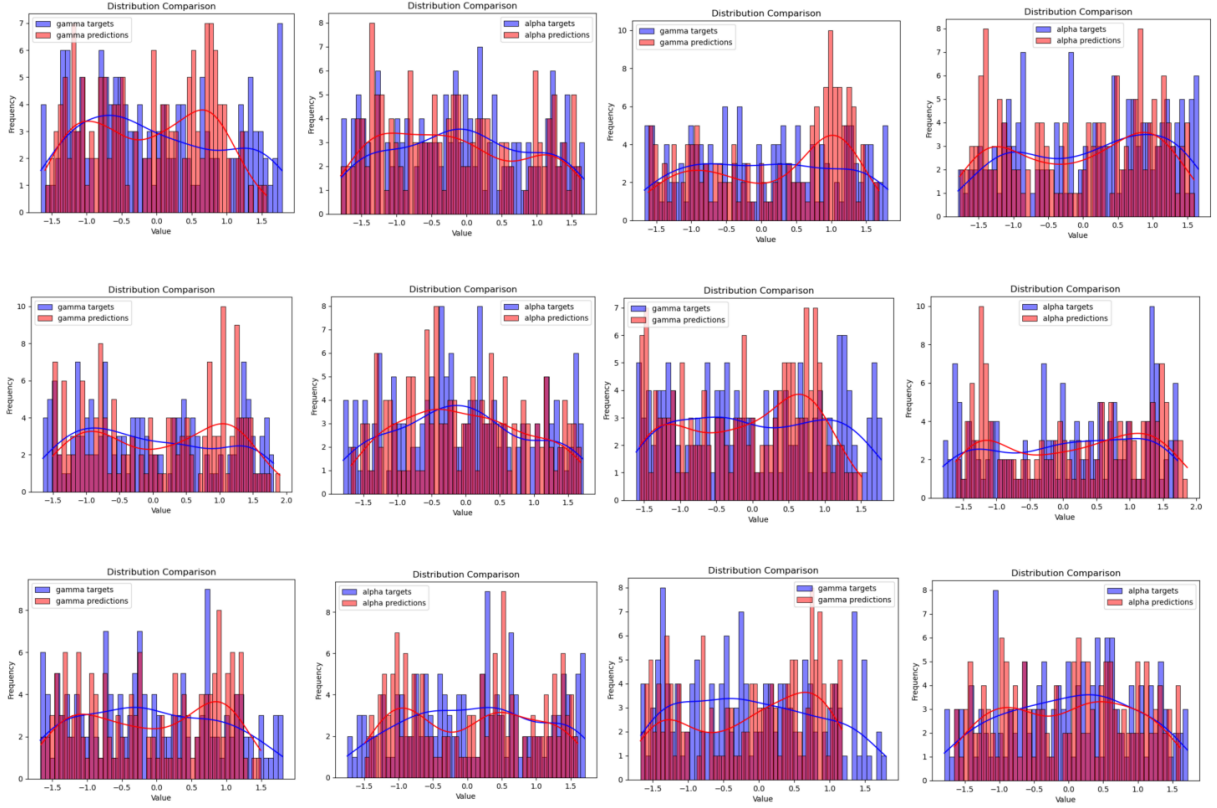


Figure 3.5: Predictions vs targets of 6 cases. The orders are 57 features (top-left), 47 features (top-right), 37 features (middle-left), 27 features (middle-right), 17 features (bottom-left), and 7 features (bottom-right).

| Number of Features | Testing MSE |
|--------------------|-------------|
| 57                 | 0.1293      |
| 47                 | 0.1149      |
| 37                 | 0.1042      |
| 27                 | 0.1017      |
| 17                 | 0.0899      |
| 7                  | 0.1544      |

Table 3.1: Testing mean squared error (MSE) for different numbers of input features.

As model complexity varies, the optimal feature selection changes, so the most effective

approach is to loop over all possible model architectures and feature sets, but this is computationally expensive. The only option we have is to fix a model architecture that makes sense for our project and then identify the best feature set for it. It does not mean that the feature set is the best overall, but it means that it is the best given the fixed model architecture. 17 features are selected because they yield the lowest MSE among the 6 selected feature sets. I applied 27 features to generate the results in this report, and obtained the results in [Table 3.1](#) afterwards. I will apply 17 features in the future, as this feature set yielded the lowest MSE.

### 3.3 Parameter Predictions

At the early stage of this project, we encountered issues with plate predictions. That means when the trained model is used to predict gamma and alpha parameters on either training or testing data, it can only predict almost the same value around the mean of gamma and alpha, because it cannot learn any useful patterns from the given samples, and the best way to lower the loss is to simply produce the target mean in this case. We eventually realized that the issue was due to a programming error that generated similar samples across different targets. This section shows that, once the programming issue is fixed, the distributions of the predicted parameters closely match the target distributions. In simple terms, this is another way to show that trained models perform well on the regression task, unlike the plate predictions observed previously.

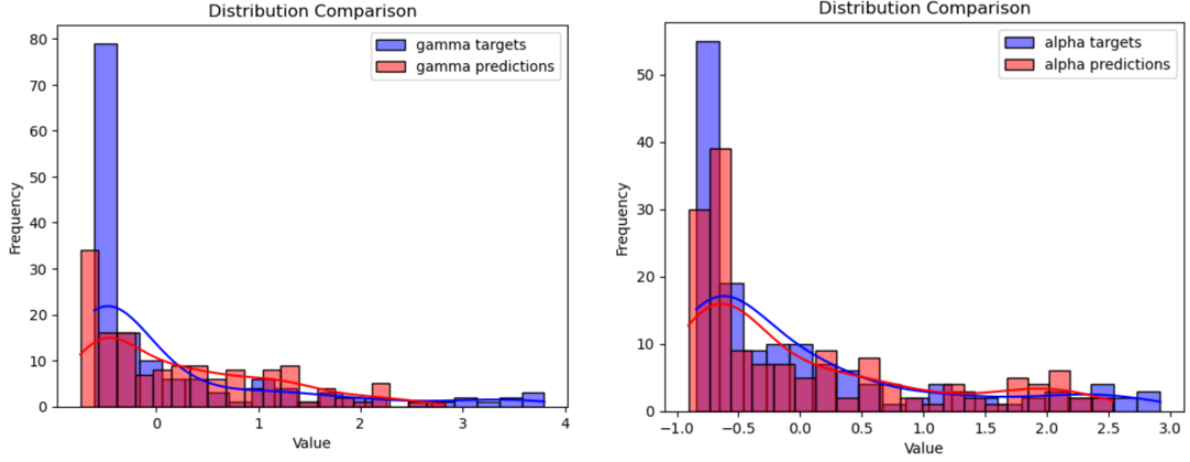


Figure 3.6: Comparison between GNN predictions and targets when the Z-score normalization is applied to both features and targets during the training. Note that targets are gamma and alpha parameters here.

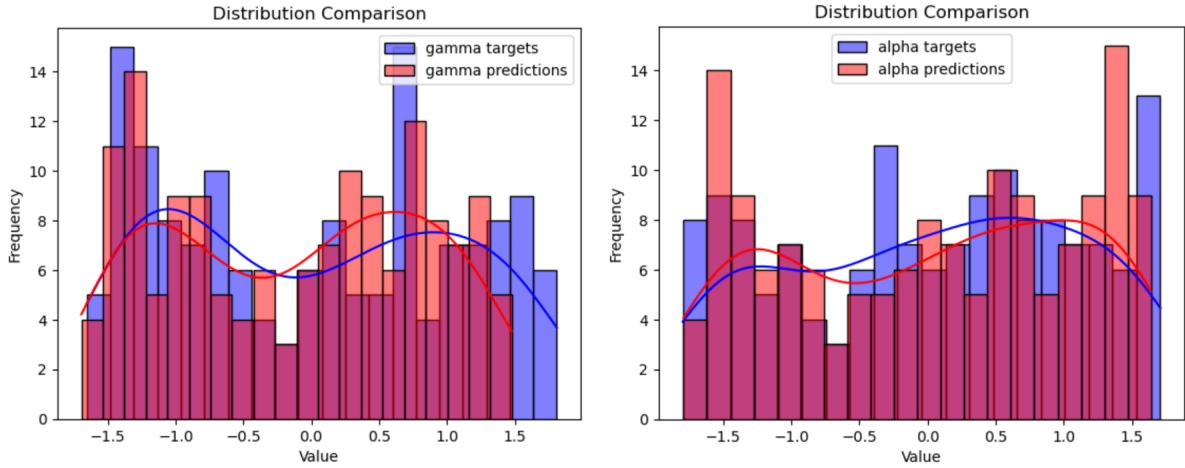


Figure 3.7: Comparison between GNN predictions and targets when the Z-score normalization is applied to both features and the logarithm base 10 of targets during the training. Note that targets are gamma and alpha parameters here.

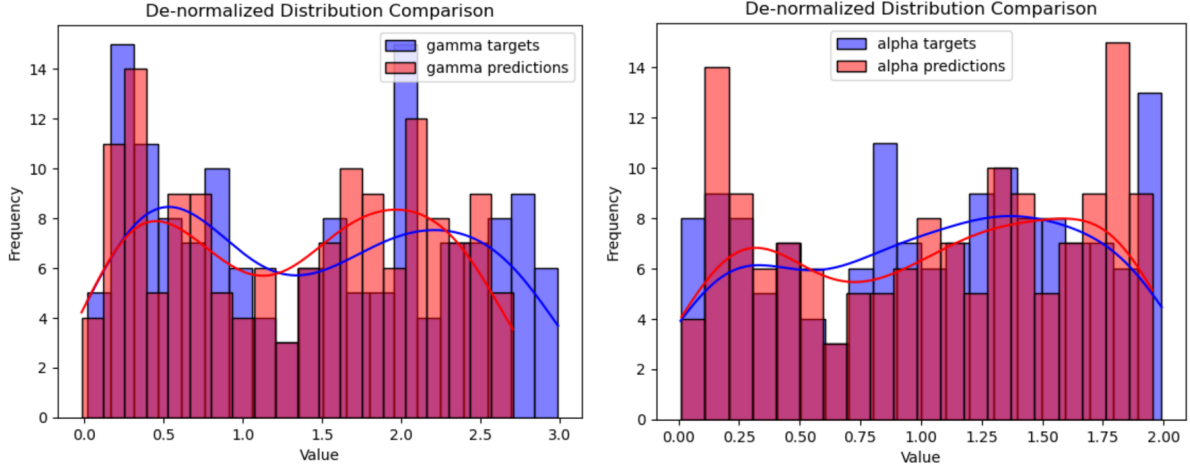


Figure 3.8: Comparison between GNN predictions and targets when the Z-score normalization is applied to both features and the logarithm base 10 of targets during the training, but the targets are de-normalized after the training. Note that targets are gamma and alpha parameters here.

All three figures show that the gamma and alpha predictions are close to their targets. As described in [subsection 2.4.2](#), the regression loss is the MSE, which measures the squared difference between predictions and targets. The close agreement with the targets indicates a low MSE, indicating good training performance. As described in [subsection 2.4.2](#) again, Z-score normalization and logarithm base 10 are applied to predictions and targets. Three figures show the same thing, but [Figure 3.7](#) shows logarithm base 10 of [Figure 3.7](#), and [Figure 3.8](#) shows de-normalized [Figure 3.7](#).

### 3.4 Data Subset Ratio

As described in [subsection 2.4.1](#), a total of 1000 simulated samples are used for training. Sometimes, due to extreme numerical issues arising from the parametrization of the simulation tool `CellModeller`, the actual number of simulated samples is between 990 and 1000. I separated these into 3 subsets for training, validation, and testing. As described in [subsection 2.4.1](#) again, the ratio is 7:1.5:1.5, and I shuffled the samples so that the target range is evenly distributed across all 3 subsets. This is one way to avoid the model-mismatch issue during training. I did a training run using the same architecture as described in [section subsection 2.4.2](#).

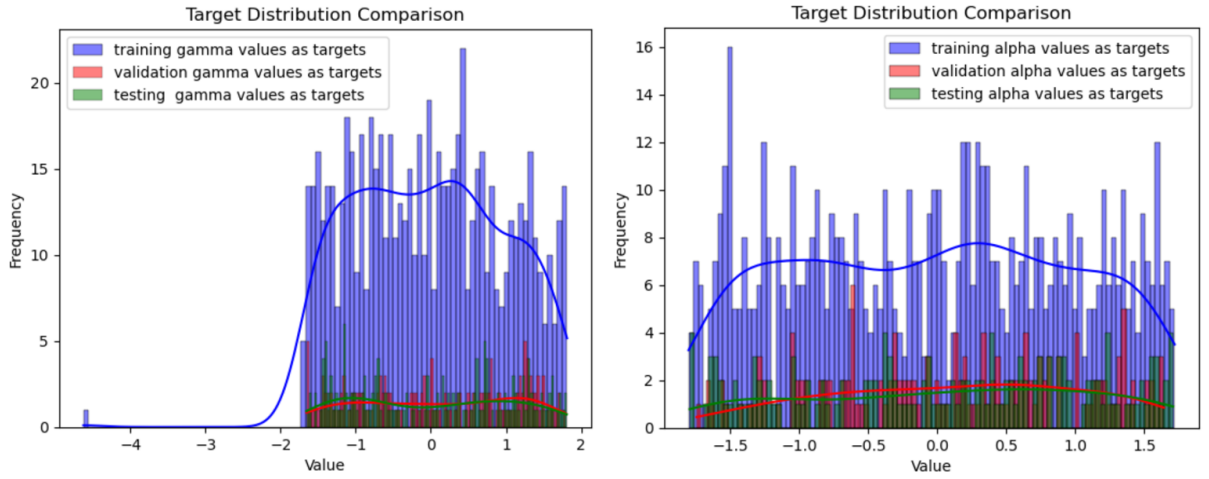


Figure 3.9: Plots to verify that the target distribution in training/validating/testing sets is close. The targets are in the log space, base  $e$ , and are normalized using Z-score normalization.

Figure 3.9 indicates that all three subsets of 1000 samples and their targets are evenly shuffled, so there is no bias to a particular gamma and alpha values to be trained. It is clear that although the training values are all above the validation and test values, the shapes are nearly identical, indicating a uniform distribution over the range  $-1.75$  to  $1.75$ . This range is caused by the Z-score normalization as described in subsection 2.4.2. Training curves are higher simply because the training set has 4 times as many values as the validation and testing sets, so the training distribution curve is 4 times higher than the validation and testing ones due to higher frequency.

## 3.5 Result Comparison between Strategies

### 3.5.1 Training Results

The strategies described in subsection 2.4.2 and subsection 2.4.3 can be compared because they use the same loss function (MSE). The strategies described in subsection 2.4.4 and subsection 2.4.5 can be compared because they use the same loss function (triplet loss). Note that a maximum of 500 epochs and early stopping with a patience of 20 are applied, as described in subsection 2.4.1.

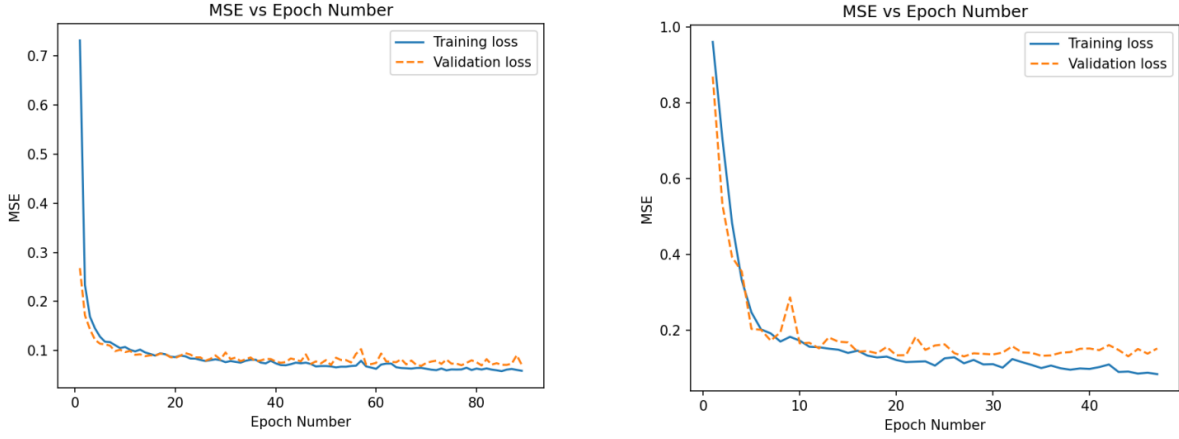


Figure 3.10: Training result comparison between GNN regressor (as described in [subsection 2.4.2](#)) and GNN-LSTM regressor (as described in [subsection 2.4.3](#)) in the ABC.

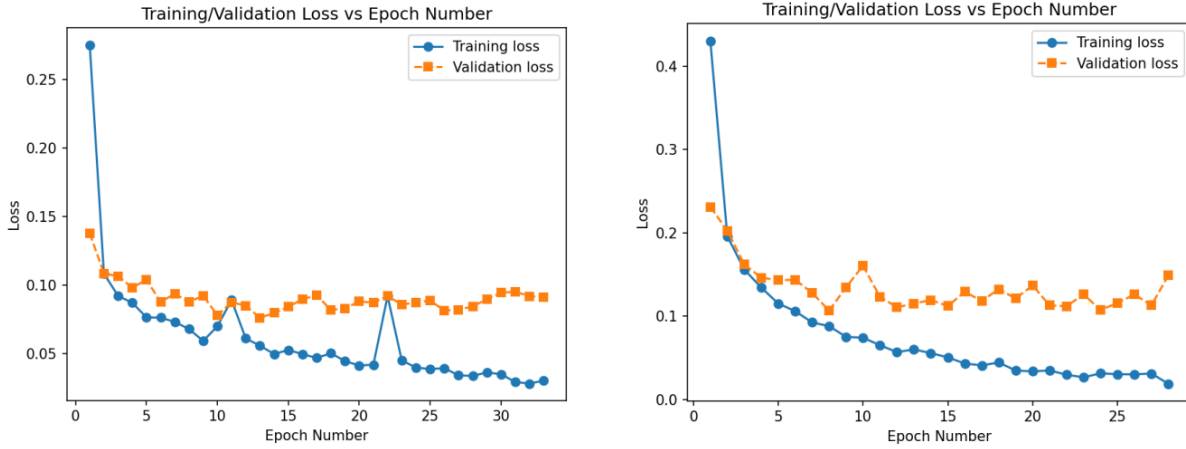


Figure 3.11: Training result comparison between GNN embedding learner (as described in [subsection 2.4.4](#)) and GNN-LSTM embedding learner (as described in [subsection 2.4.5](#)) in the ABC.

The left training result is more stable and provides lower MSE values than the right training result in [Figure 3.10](#). The left training result provides lower MSE values than the right training result in [Figure 3.11](#). It means that the GNN regressor (as described in [subsection 2.4.2](#)) is better than the GNN-LSTM regressor (as described in [subsection 2.4.3](#)),

and the GNN embedding learner (as described in [subsection 2.4.4](#)) is better than the GNN-LSTM embedding learner (as described in [subsection 2.4.5](#)) in the ABC. This means that the lineage edge type can handle temporal information more effectively than LSTM mechanisms. Training a knowledge graph not only saves time but also generates better results.

All four GNN-based strategies have good model performance. As described in [section 1.7](#), the better the model performance, the more informative its summary statistics will be. This section shows that all GNN-based ABCs achieve good model performance and that their associated GNN-based summary statistics  $S_\phi$  are informative. However, we cannot compare them to determine which strategy yields the highest informativeness, because they use different training loss functions and optimization mechanisms. An efficient way to compare them is to establish a ground-truth baseline and apply simulation-based validation as described in [subsection 2.5.5](#) and [subsection 2.5.3](#).

### 3.5.2 Posterior Inference Result Comparisons between Strategies

We evaluate the parameter inference results by determining the KL divergence (as described in [subsection 2.5.2](#)) and the MSE (described in [subsection 2.5.6](#)) between the last inferred posterior and the Gaussian approximation of a Dirac delta function for each strategy.

Note that [Figure 2.11](#) in [subsection 2.5.2](#) shows how each selected  $\varepsilon$  forms the Gaussian distribution to approximate the Dirac delta functions. Note that  $\varepsilon$  stands for the thickness of the approximated delta function. The lower  $\varepsilon$  is, the closer it is to the ideal delta function. Please refer to this figure to visualize the approximations.

Table 3.2: Simulation-based validation results for strategies when  $\varepsilon = 0.02$ .

| Pipeline     | Gamma KL  | Alpha KL | Joint KL  | MSE     |
|--------------|-----------|----------|-----------|---------|
| 1st Strategy | 25.31857  | 0.91652  | 26.30186  | 0.02389 |
| 2nd Strategy | 57.06778  | 1.12551  | 58.26087  | 0.04962 |
| 3rd Strategy | 38.12156  | 1.37422  | 41.29812  | 0.03472 |
| 4th Strategy | 39.69418  | 1.43704  | 41.14441  | 0.03598 |
| 5th Strategy | 103.89214 | 12.24659 | 116.15056 | 0.09694 |

I selected  $\varepsilon = 0.02$  because lower values are better, and the approximation fails for  $\varepsilon < 0.2$ . The joint KL divergence equals the sum of marginal KL divergences if the

parameters are independent of each other. [Table 3.2](#) shows that the sum of marginal KL divergences is very close to the associated joint KL divergence, so gamma and alpha are likely independent parameters as how we set them up in [section 1.5](#).

[Table 3.2](#) shows that the GNN regressor performs the best over all strategies, and all GNN-based strategies are better than the conventional method. The lower the KL divergence and the MSE, the better the inference result. It means that all four GNN-based ABCs generate more informative summary statistics than the conventional method, as described in [subsection 2.5.5](#).

Thus, if there is no model mismatch, all four GNN-based strategies (as described in [subsection 2.4.2](#), [subsection 2.4.3](#), [subsection 2.4.4](#), [subsection 2.4.5](#)) outperform the conventional method (as described in [subsection 2.4.6](#)), which requires manual selection of summary statistics using expert knowledge. Moreover, the GNN regressor (as described in [subsection 2.4.2](#)) outperforms the other GNN-based strategies.

### 3.5.3 Posterior Inference Results for Strategies

As described in [section 2.3](#), I chose

$$\gamma = 200 \tag{3.1}$$

$$\alpha = 35 \tag{3.2}$$

$$\ln(\gamma) \approx 2.301 \tag{3.3}$$

$$\ln(\alpha) \approx 1.544 \tag{3.4}$$

All the strategies as described in [subsection 2.4.1](#) are applied to the ABC-SMC pipeline, and the parameter posterior inference results are over generations as described in [section 2.3](#) for each strategy.

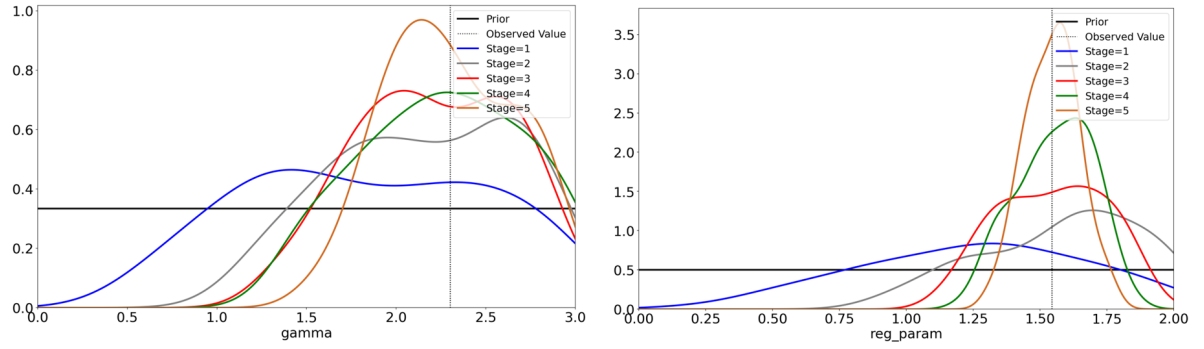


Figure 3.12: ABC-SMC posterior results over generations for GNN regressor (as described in [subsection 2.4.2](#)).

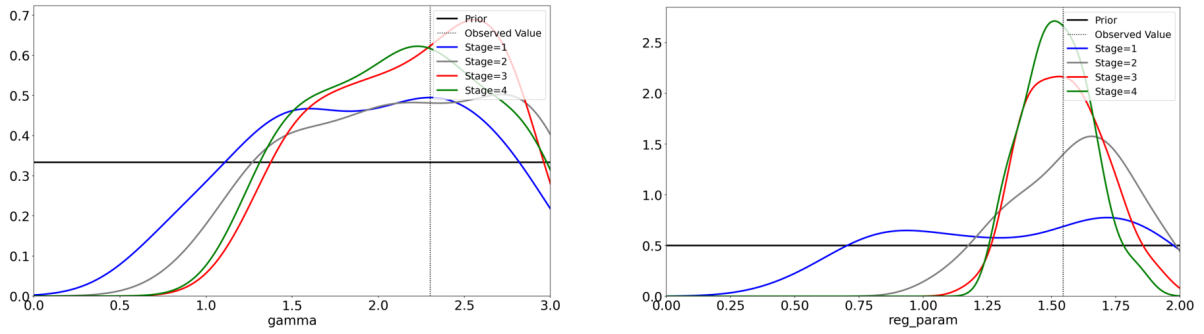


Figure 3.13: ABC-SMC posterior results over generations for GNN-LSTM regressor (as described in [subsection 2.4.3](#)).

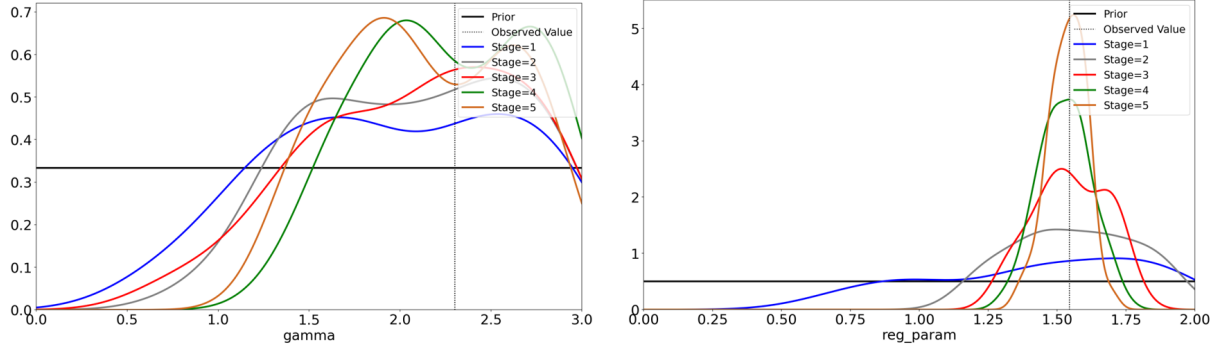


Figure 3.14: ABC-SMC posterior results over generations for GNN embedding learner (as described in [subsection 2.4.4](#)).

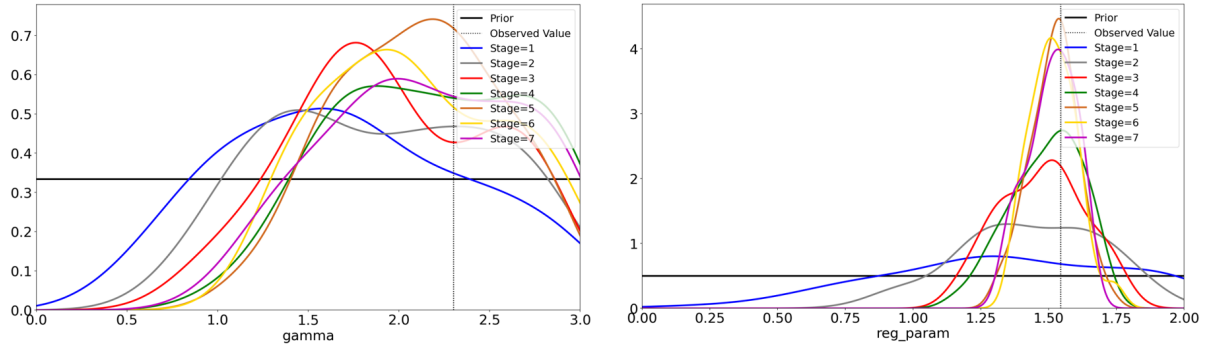


Figure 3.15: ABC-SMC posterior results over generations for GNN-LSTM embedding learner (as described in [subsection 2.4.5](#)).

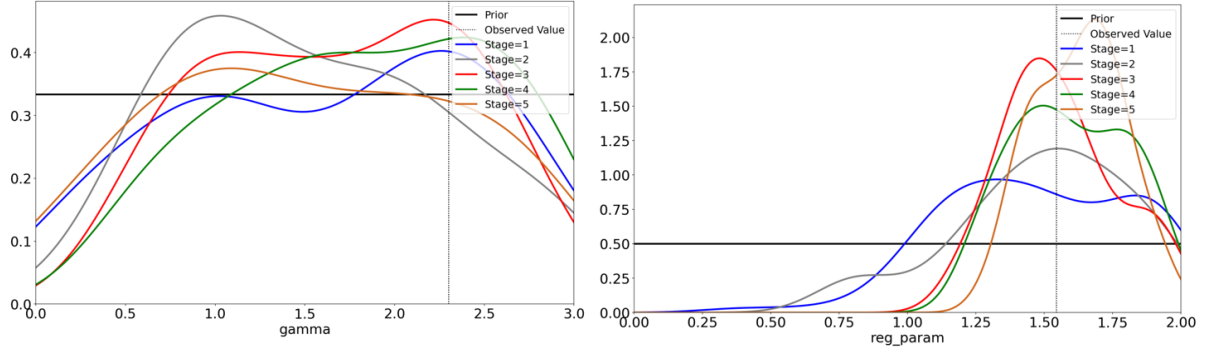


Figure 3.16: ABC-SMC posterior results over generations for the conventional method (as described in [subsection 2.4.6](#)).

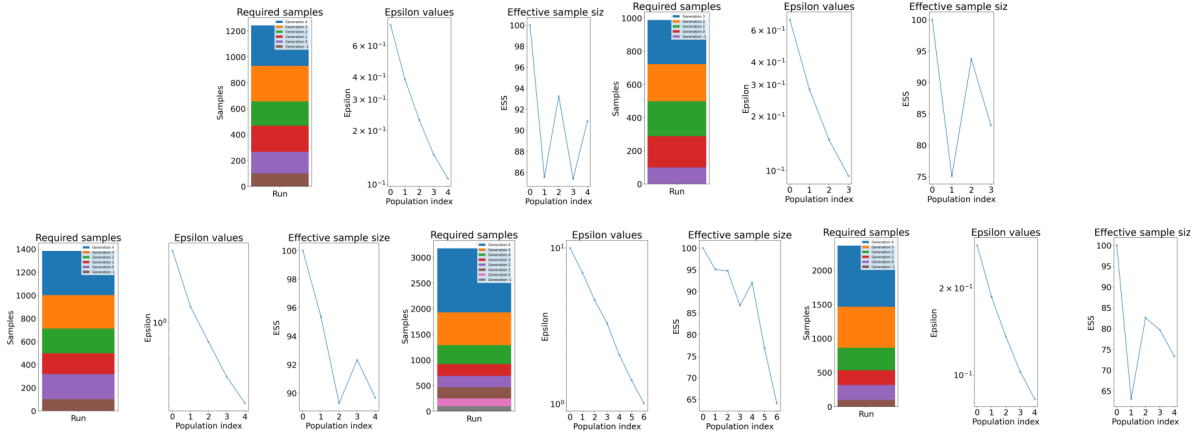


Figure 3.17: More helpful ABC-SMC information.

All figures in this section are intended to illustrate the process of how the posterior inference gets done over generations for each strategy. This is to observe the intermediate posterior results, except for the last generation.

The left posterior distributions in each figure are the result of parameter gamma. If we compare the results of left posterior inference across all five figures above, we see that [Figure 3.12](#) has the highest peak, and its results improve over generations. The other four figures show that their left posterior results improve over a couple of generations, then worsen and stop improving.

The right posterior distributions in each figure are the result of parameter alpha. If we compare the results of left posterior inference across all five figures above, we see that [Figure 3.14](#) has the highest peak, and all figures show that their results improve over generations.

### 3.6 Conventional Inference Result with Observations of Real Experiments

The main goal of this section is to reproduce the result of the previous case study (in [Appendix B](#)) done by my seniors, Atiyeh Ahmadi and Aaron Yip, in our research group [2]. As described in [subsection 2.4.1](#), I chose to replace the logarithm base 10 with the logarithm base e to reproduce the inference result of the paper, since the paper applies the logarithm base e. Note that the ABC-SMC is applied to real-experimental data, rather than simulated-experimental data. The sampling validation is applied as well, as described in [subsection 2.5.7](#).

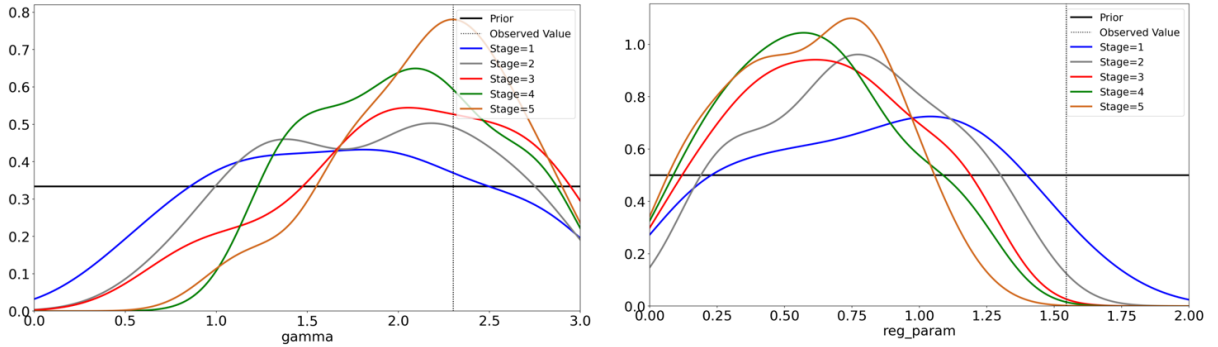


Figure 3.18: ABC-SMC posterior results over generations, with logarithm base 10 and relative distance.

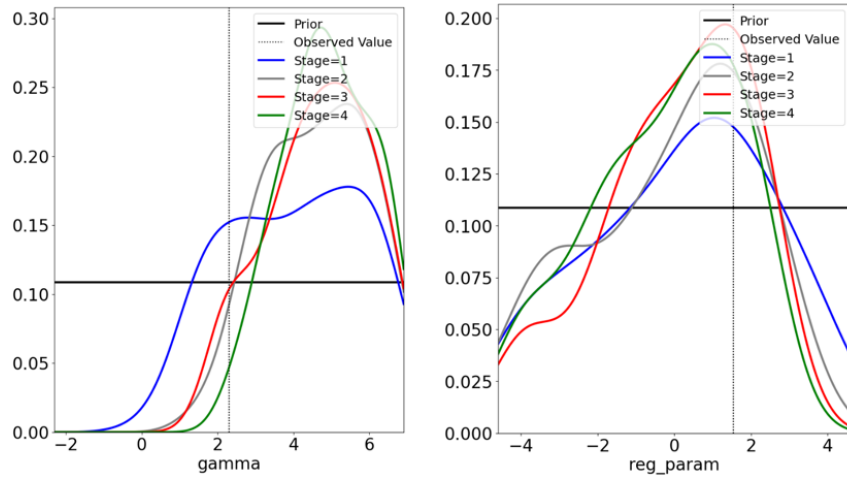


Figure 3.19: ABC-SMC posterior results over generations, with logarithm base e and relative distance.

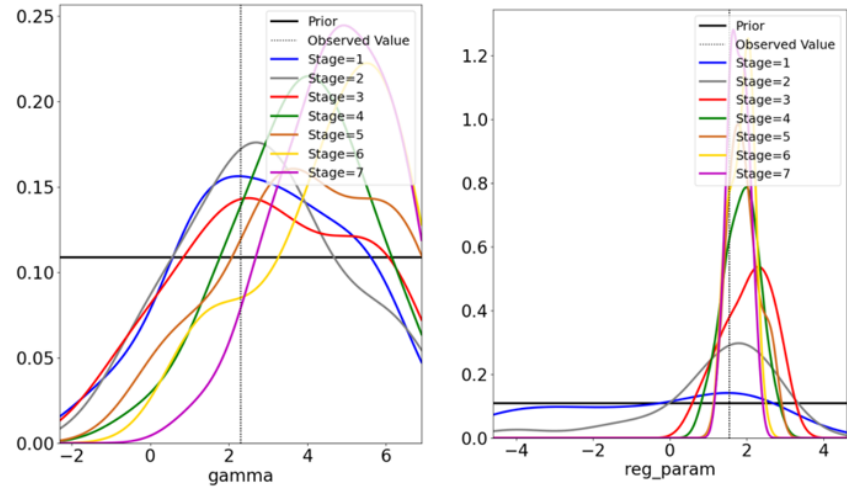


Figure 3.20: ABC-SMC posterior results over generations, with logarithm base e and KL divergence.

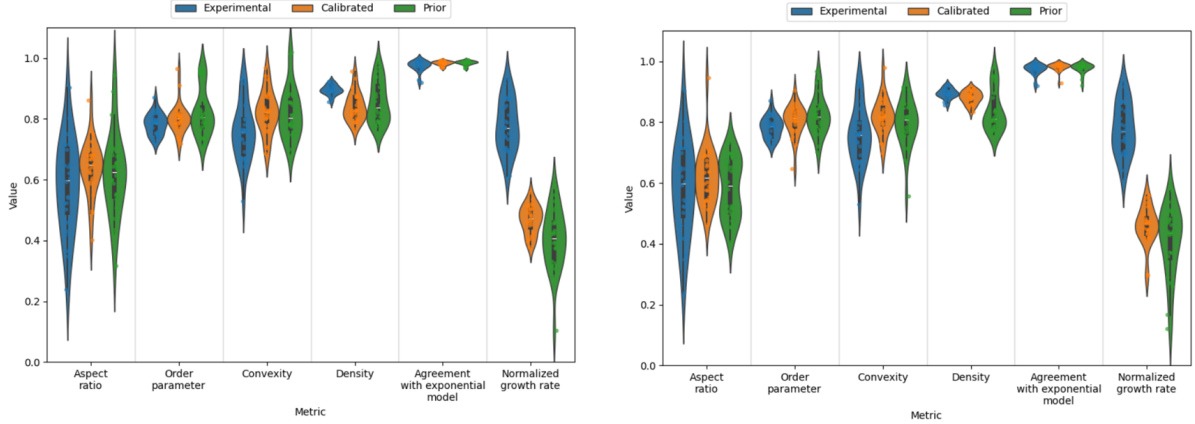


Figure 3.21: ABC-SMC posterior sampling validation, with logarithm base  $e$ . The left result is based on the relative distance. The right result is based on the KL divergence.

When the relative distance is replaced with the KL divergence, the alpha result improves, but the gamma result stays the same. The possible reason is that gamma is less related to the features of the training set than alpha. The general trends in both results are nearly identical. The result is exactly the same as that of the paper [2].

### 3.7 Graph Neural Network-based Inference Result with Observations of Real Experiments

As shown in [subsection 3.5.2](#), when I applied the simulated-experimental data to test the inference pipeline performance, it is guaranteed that there is no model mismatch at all, since both the simulated-experimental data and the training data are generated by `CellModeller`. As shown in [Table 3.2](#), the posterior results indicate that all four GNN-based strategies outperform the conventional method and that the GNN regression on knowledge graphs outperforms the other three GNN-based strategies. However, when simulated-experimental data are replaced with real-experimental data for testing, the model mismatch issue leads to poor inference results, as follows.

We used the trained GNN regression model  $S_\phi$  described in [subsection 2.4.2](#) to predict on 24 observations  $y$  and obtain the predictions  $S_\phi(y)$ . Note that the model  $S_\phi$  was trained using simulated data  $x$ . If  $x$  and  $y$  are from very different populations,  $S_\phi$  can only be

informative to  $x$  but not to  $y$  due to the model mismatch. Note that the performance of this trained GNN regression model can be referred back to [Figure 3.10](#) in [subsection 3.5.1](#).

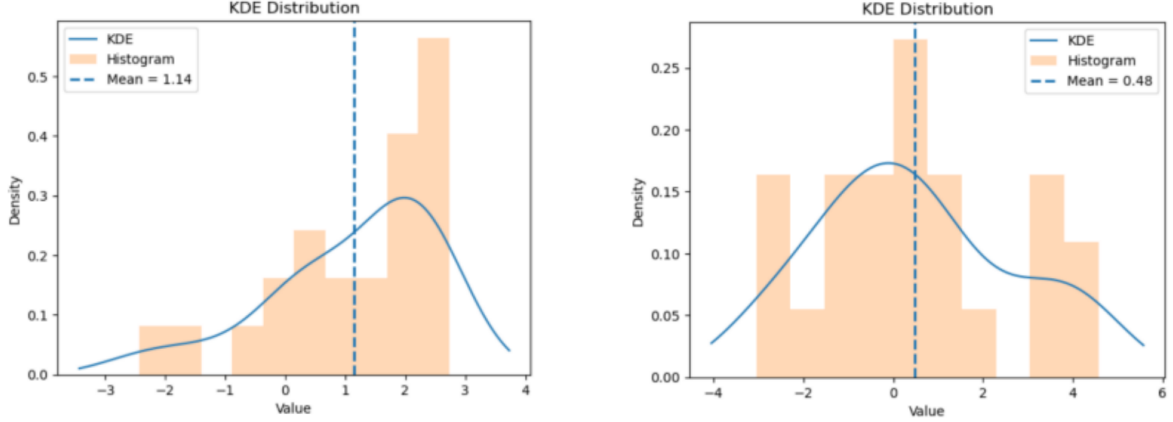


Figure 3.22: GNN regression predictions on 24 observations of real experiments. Note that the parameters are in the scale of log base  $e$ . The left figure is for parameter  $\gamma$  and the right figure is for parameter  $\alpha$ .

We then used the trained GNN regression model and 24 observations from real experiments to run the ABC pipeline. We applied the relative  $\ell_1$  distance ([Equation 1.230](#)) and the KL divergence ([Equation 1.241](#)) as described in [subsection 1.6.5](#), respectively.

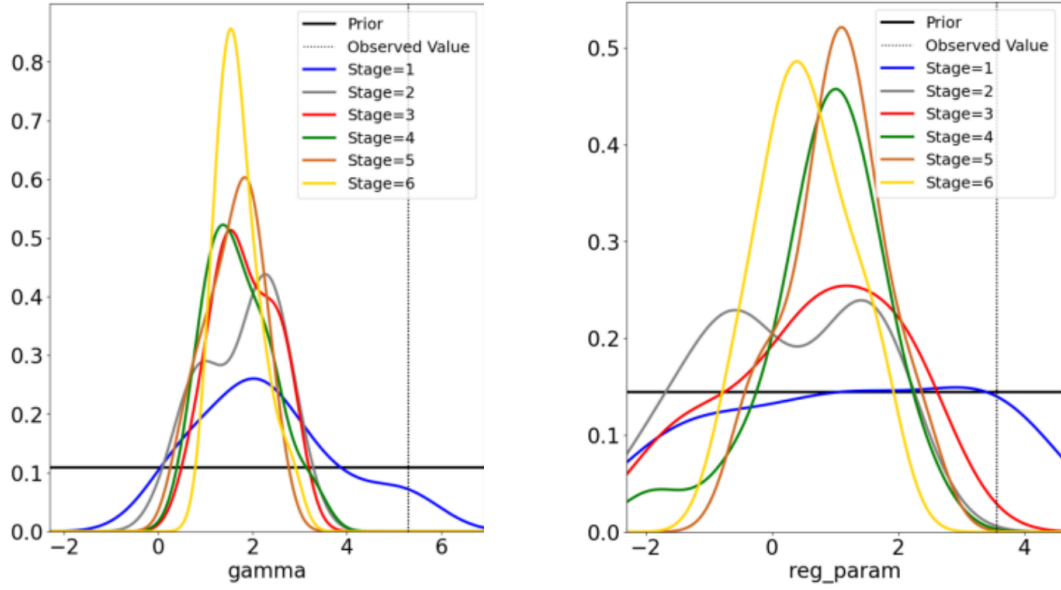


Figure 3.23: GNN regression ABC inference results on 24 observations of real experiments using the relative  $\ell_1$  distance. Note that the parameters are in the scale of log base  $e$ . The left figure is for parameter  $\gamma$  and the right figure is for parameter  $\alpha$ .

Figure 3.23 shows similar patterns as Figure 3.22, because it takes the mean values of Figure 3.22 as the information to calculate the distance metric in ABC.

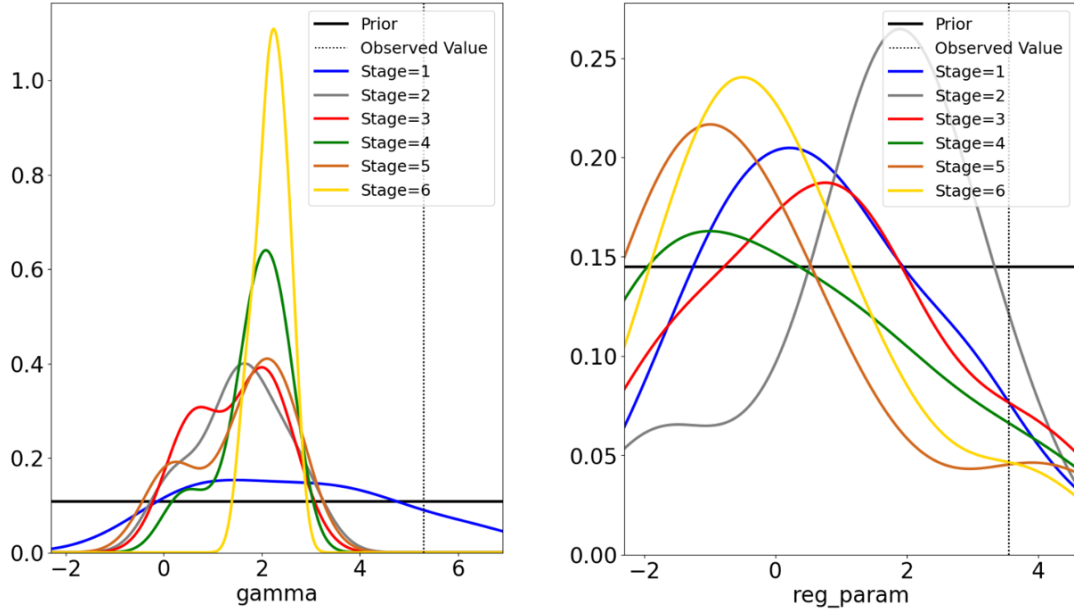


Figure 3.24: GNN regression ABC inference results on 24 observations of real experiments using the KL divergence. Note that the parameters are in the scale of log base  $e$ . The left figure is for parameter  $\gamma$  and the right figure is for parameter  $\alpha$ .

Figure 3.23 and Figure 3.24 show different posterior results from those of Figure 3.19 and Figure 3.20. Figure 3.24 shows exactly the same patterns as Figure 3.22, even closer to it than Figure 3.23.

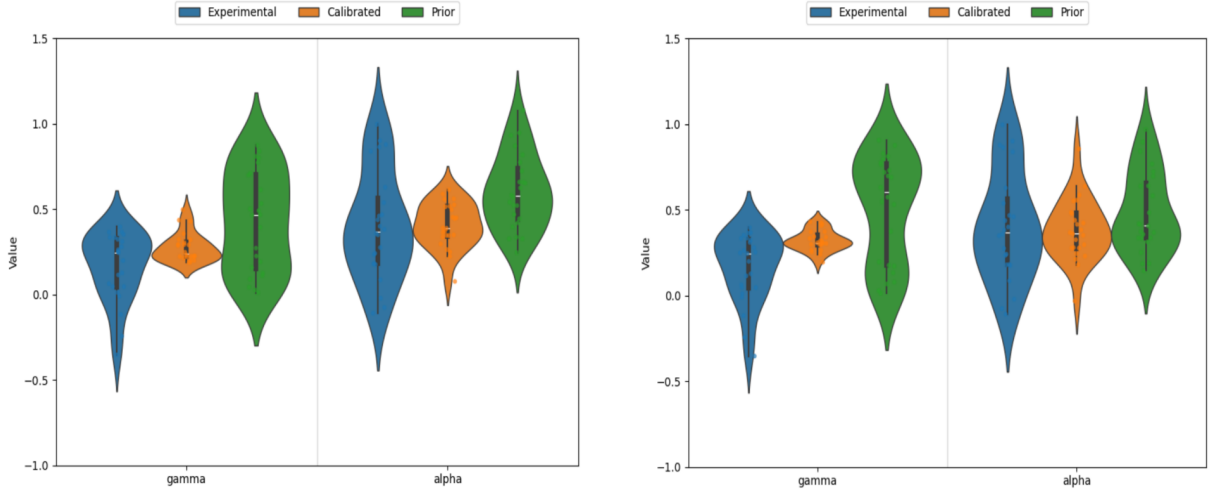


Figure 3.25: Sampling validation result of GNN regression ABC inference. The left figure uses the relative distance as the distance function in ABC, and the right figure uses the KL divergence as the distance function in ABC. Note that the parameters are normalized. In each figure, the left parameter is  $\gamma$  and the right parameter is  $\alpha$ .

Figure 3.25 shows the sampling validation result, which we can use to compare with the conventional approach in Figure 3.21 (or Figure B.3), but here the summary statistics are the parameters themselves. The posterior sampling result is very close to the experimental sampling result, with a much lower variance. The posterior differs substantially from the prior, indicating that the summary statistics are informative in some sense.

### 3.8 Issue with Node Connectivity in Graphs and Model Discrepancy

The connectivity of the simulated graphs differs substantially from that of the experimental graphs. This is because our pipeline could not identify the correct connectivity due to model discrepancies arising from overly high parameter values.

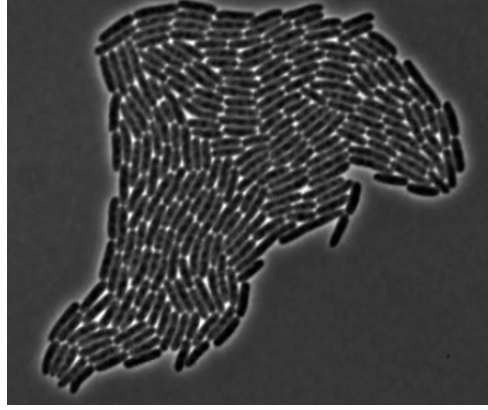


Figure 3.26: Experimental image of the last time point. This is from one of 24 experimental samples. As described in [section 1.4](#), it was taken by my senior, Atiyeh Ahmadi [2].

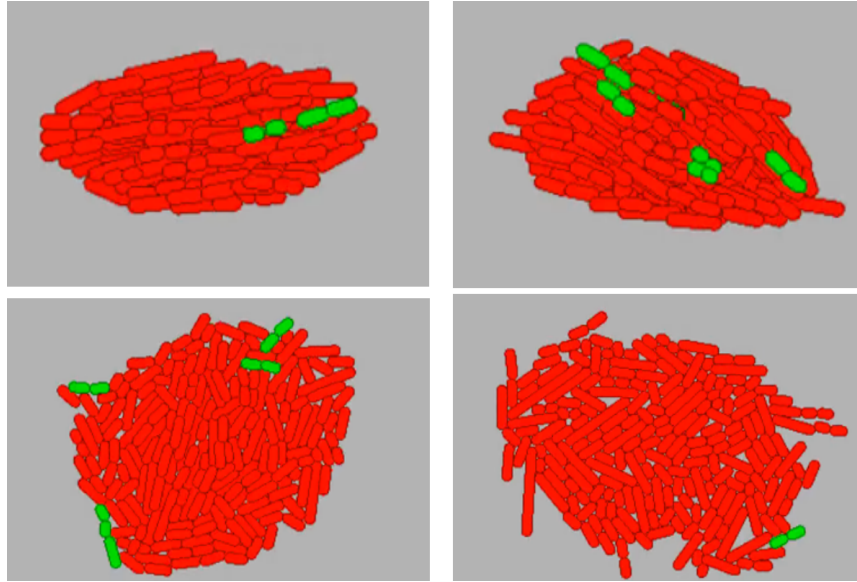


Figure 3.27: Simulated images of the last time point. These 4 images are from 4 simulated samples given different parameter values:  $\gamma = 1$ ,  $\alpha = 50$  (top-left);  $\gamma = 500$ ,  $\alpha = 50$  (top-right);  $\gamma = 500$ ,  $\alpha = 0.5$  (bottom-left);  $\gamma = 1$ ,  $\alpha = 0.5$  (bottom-right).

As described in [subsection 2.1.2](#), the parameter ranges of the training set and the prior in ABC are

$$\gamma \sim \text{Loguniform}(0.1, 1000), \quad (3.5)$$

$$\alpha \sim \text{Loguniform}(0.1, 100). \quad (3.6)$$

These ranges were defined because we supposed that they were biologically realistic, but they are not.

Figure 3.26 is the real experimental image. This is from one of the experimental samples, but all samples should exhibit similar behavior. Our pipeline identified approximately 14,000 edges between nodes (i.e., cells) for Figure 3.26. However, our pipeline identified fewer than 1000 edges for some simulated samples.

Figure 3.27 shows 4 simulated samples given 4 pairs of parameter values. Our pipeline identified 900 edges in the top-left figure, 1200 in the top-right figure, 11000 in the bottom-left figure, and 15000 in the bottom-right figure. For some reason, our pipeline was unable to correctly identify the number of edges when the nodes overlap heavily. Note that we set up the simulation module to handle only 2-dimensional cases and turned off 3-dimensional cases in this project, so the overlapping cells cause the model discrepancy because they are not meaningful.

Figure 3.26 seems to have low  $\gamma$  and low  $\alpha$  values. Figure 3.27 shows that  $\alpha$  affects the issue of cell overlap much more than  $\gamma$ . To fix this, a simple and straightforward solution is to narrow the parameter range we explore, including the training set and prior in ABC. For example, we can set things up as

$$\gamma \sim \text{Loguniform}(0.1, 100), \quad (3.7)$$

$$\alpha \sim \text{Loguniform}(0.1, 10), \quad (3.8)$$

and try to run the ABC inference pipeline again. Note that we need to be cautious about over-confident posterior estimation due to an overly small acceptance threshold  $\varepsilon$ .

## 3.9 Future Work

In summary, there are 2 remaining issues in this project: addressing the model discrepancy due to connectivity (as described in section 3.8) and handling the general model mismatch.

In terms of general model mismatch, several sources can cause such an issue.

1. In `CellModeller`, some parameters are defaults, and some parameters are specified based on observations from experiments. Recall that these specified parameters are called observable parameters. We can check whether these parameters are consistent with the experimental results.

2. In **CellModeller**, some unexplored parameters may exist in real experiments, but we do not know their values or how many. This requires further development within the **CellModeller** package to improve it.
3. In terms of simulated samples, the simulated pickle files are originally processed by some output processing scripts. It is important to verify that all parameters and features are expressed in the same units.

Note that **TrackRefiner** does not cause a model mismatch in this pipeline because it does not generate new content, but it can cause errors.

I compared the **CellModeller** parameter distributions between simulations and experiments. The parameters are: cell radius, division length threshold, pickle step as described in [section 1.5](#).

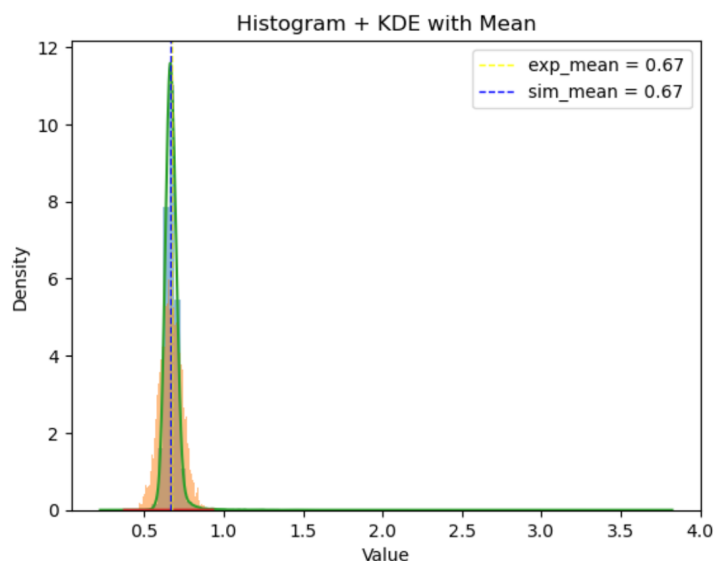


Figure 3.28: Cell radius parameter distribution comparison between simulations and experiments. The cell radius is one of the specified observable parameters in the **CellModeller** simulation module.

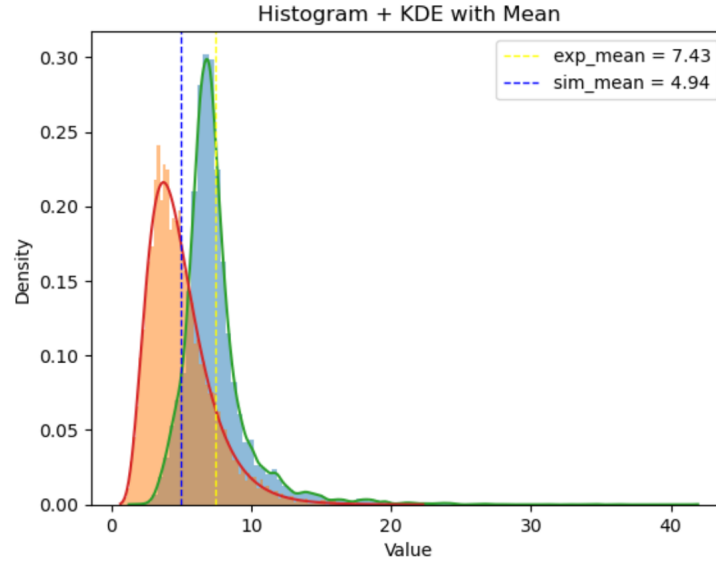


Figure 3.29: Division length threshold parameter distribution comparison between simulations and experiments. The division length threshold is one of the specified observable parameters in the `CellModeller` simulation module.

From [Figure 3.28](#) and [Figure 3.29](#), we see that the simulation setup for the cell radius and division length threshold parameters is very close to the experimental samples. This means that the source of the mismatch is not in these two parameters, and we can leave them as is. This is a good example of a match.

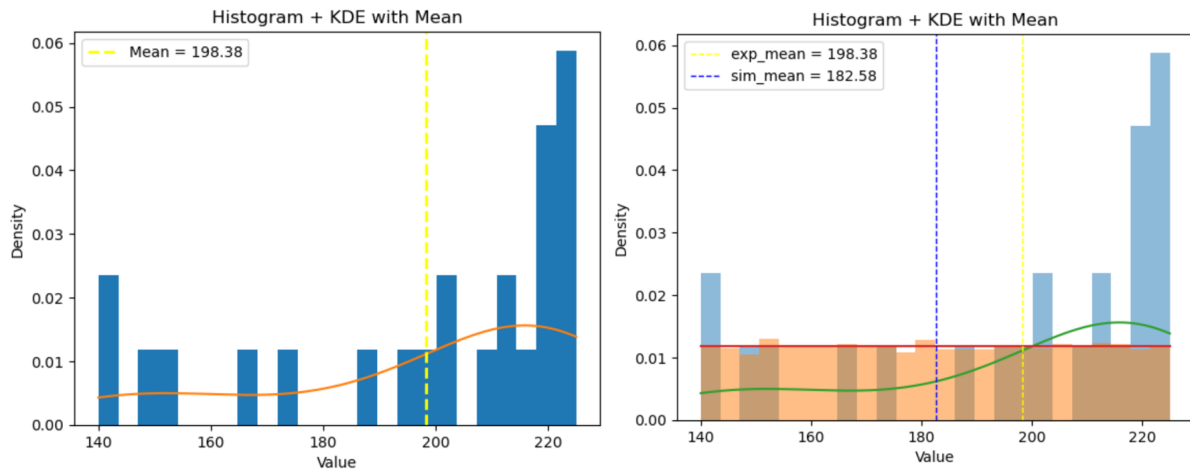


Figure 3.30: Pickle step parameter distribution comparison between simulations and experiments. The pickle step parameter specifies the number of time steps between two records, where each time step is 0.025 seconds. The left figure shows the experimental distribution. The right figure shows both the experimental and simulated distributions when the random selection is applied to the pickle step.

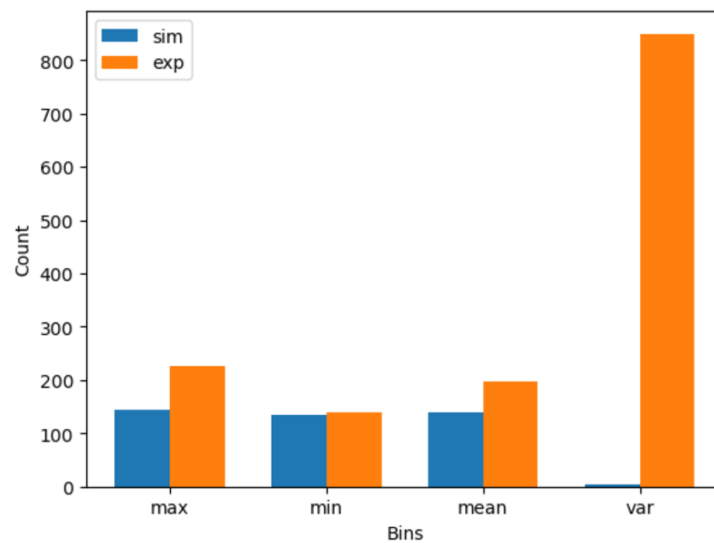


Figure 3.31: Pickle step parameter distribution comparison between simulations and experiments. This simulated pickle step is specified as a deterministic value 140.

From [Figure 3.30](#) and [Figure 3.31](#), we see that our previous setup for the pickle step parameter deviates substantially from the experimental data. If the previous deterministic pickle step value is replaced with a uniform distribution that randomly selects the pickle step value, the resulting distribution becomes much closer to the experimental samples. This is a good example of a mismatch.

As long as the graph sizes and structures in the simulated samples are very close to those in the experimental samples, the calibration result should be sufficient for sampling validation. The actual gamma and alpha parameter distributions are unknown in experiments, so although some unexplored parameters still cause model mismatch, the calibration pipeline still serves to bridge simulations and experiments by focusing on gamma and alpha only.

Several approaches to solve the model mismatch were proposed:

1. Average feature means and variances between simulations and experiments, and apply to Z-score normalization of features as input to trained GNN models.
2. Incorporate experimental data more directly into training.
3. Do preliminary training on less relevant experimental data.

Note that we proposed comparing feature distributions between experiments and simulations, but the simulated data are generated based on the prior. Thus, this doesn't make sense.

Although this posterior inference result appears to be affected by model mismatch to a considerable extent, the good training performance and simulation-based validation results directly demonstrate that GNN-based summary statistics are highly informative, even more informative than conventional summary statistics. Even if we can resolve this model mismatch, applying this pipeline to other cases will introduce new mismatches, and we will never know how severe the next mismatch will be.

## 3.10 Conclusion

In this study, four strategies were developed to integrate the ABC-SMC framework with GNNs. The proposed approaches replace conventional handcrafted summary statistics with graph regression predictions or graph embedding representations learned by GNNs. Within this framework, the ABC-SMC algorithm was used for parameter inference, using

its adaptive acceptance threshold  $\epsilon$  and weighted particle population to efficiently calibrate agent-based models of microbiological population dynamics in growing cells.

The proposed GNN-based strategies differ in their learning objectives and mechanisms for incorporating temporal information from population dynamics. Specifically, the strategies employ different learning formulations, as reflected in the choice of loss functions, and explore multiple approaches to capture temporal dependencies. These include architectures augmented with LSTM layers, as well as graph representations that encode temporal relationships via lineage edge types in knowledge graphs.

The performance of the proposed GNN-based ABC pipelines was evaluated and compared with that of a conventional approach using manually defined summary statistics. The evaluation criteria were the KL divergence between posterior distributions and Gaussian approximations of Dirac delta functions. Among the four GNN-based strategies, the first strategy achieved the best performance. This strategy used a graph regression model comprising four GAT layers with two attention heads and two edge types, replacing conventional summary statistics with parameter predictions obtained from the trained GNN regression model.

The results demonstrate that integrating ABC-SMC with GNNs, including models augmented with LSTMs, yields improved calibration performance compared to the conventional approach, assuming no model mismatch and no unknown noise from observations of real experiments. The most promising strategy (i.e., GNN regression) was incorporated into the ABC pipeline using real experimental observations, rather than the simulated experimental data used as the ground-truth baseline. GNN-based architectures may not handle model mismatch well due to the over-reliance on simulated data, and more plausible solutions are considered for future work. In general, the proposed methodology highlights the potential of combining computational Bayesian inference with deep learning to obtain informative summary statistics and to enable efficient parameter estimation in complex mechanistic models.

# References

- [1] A. Ahmadi, A. Dostmohammadi, R. Mclean, and B. Ingalls. Trackrefiner: a tool for refinement of bacillus cell tracking data. *npj Systems Biology and Applications*, 11(1):127, 2025.
- [2] Atiyeh Ahmadi, Aaron Yip, Jonathan Chalaturnyk, Marc G. Aucoin, and Brian P. Ingalls. A pipeline for calibrating agent-based models of microbial populations: From image collection to model parameterization. *IFAC-PapersOnLine*, 58(23):1–6, 2024.
- [3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations (ICLR)*, 2015.
- [4] Mark A. Beaumont. Approximate bayesian computation in evolution and ecology. *Annual Review of Ecology, Evolution, and Systematics*, 41:379–406, 2010.
- [5] Mark A Beaumont, Wenyang Zhang, and David J Balding. Approximate bayesian computation in population genetics. *Genetics*, 162(4):2025–2035, 2002.
- [6] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
- [7] Michael G B Blum. Approximate bayesian computation: a nonparametric perspective. *Journal of the American Statistical Association*, 105(491):1178–1187, 2010.
- [8] Michael GB Blum, Maria A Nunes, Dennis Prangle, and Scott A Sisson. A comparative review of dimension reduction methods in approximate bayesian computation. *Statistical Science*, 28(2):189–208, 2013.
- [9] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 2018.

- [10] Jianfei Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations (ICLR)*, 2018.
- [11] Oana-T. Chis, Julio R. Banga, and Eva Balsa-Canto. Structural identifiability of systems biology models: A critical comparison of methods. *PLoS ONE*, 6(11):e27755, 2011.
- [12] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *International Conference on Learning Representations (ICLR)*, 2016.
- [13] Sushma Dhamodharan. Unveiling bacterial microcolony features through graph attention representation learning. Technical report, University of Waterloo, Computational Mathematics, 2024. Master’s research project.
- [14] Paul A. M. Dirac. *The Principles of Quantum Mechanics*. Oxford University Press, 1930.
- [15] Arnaud Doucet, Nando de Freitas, and Neil Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, New York, 2001.
- [16] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.
- [17] Paul Fearnhead and Dennis Prangle. Constructing summary statistics for approximate bayesian computation. *Journal of the Royal Statistical Society: Series B*, 74(3):419–474, 2012.
- [18] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with pytorch geometric. In *ICLR Workshop*, 2019.
- [19] Andrew Gelman et al. *Bayesian Data Analysis*. CRC Press, 2013.
- [20] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of ICML*, 2017.
- [21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [22] Volker Grimm, Eloy Revilla, Uta Berger, Florian Jeltsch, Wolf M. Mooij, Steven F. Railsback, Hans-Hermann Thulke, Jacob Weiner, Thorsten Wiegand, and Donald L. DeAngelis. Pattern-oriented modeling of agent-based complex systems: lessons from ecology. *Science*, 310(5750):987–991, 2005.

- [23] William L. Hamilton. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159, 2020.
- [24] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
- [25] Zaynab Hammoud and Frank Kramer. Multilayer networks: aspects, implementations, and application in biomedicine. *Big Data Analytics*, 5, 2020.
- [26] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- [27] E. H. Hssayni, A. Boufssasse, N. E. Joudar, et al. Novel dropout approach for mitigating over-smoothing in graph neural networks. *Applied Intelligence*, 55:354, 2025.
- [28] Bai Jiang, Tung-Yu Wu, Charles Zheng, and Wing Hung Wong. Learning summary statistic for approximate bayesian computation via deep neural network. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
- [29] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.
- [30] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.
- [31] Hiroaki Kitano. Systems biology: a brief overview. *Science*, 295(5560):1662–1664, 2002.
- [32] Emmanuel Klinger, David Rickert, and Jan Hasenauer. pyabc: Distributed, likelihood-free inference. *Bioinformatics*, 34(20):3591–3593, 2018.
- [33] Edda Klipp et al. *Systems Biology: A Textbook*. Wiley, 2016.
- [34] Solomon Kullback and Richard A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.
- [35] Solomon Kullback and Richard A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 1951.

- [36] Chao Li, Xiaokong Ma, Bing Jiang, Xiangang Li, Xuewei Zhang, Xiao Liu, Ying Cao, Ajay Kannan, and Zhenyao Zhu. Deep speaker: An end-to-end neural speaker embedding system. In *Proceedings of the International Conference on Neural Information Processing (ICONIP)*, 2017.
- [37] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *Proceedings of AAAI*, 2018.
- [38] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of the 30th International Conference on Machine Learning (ICML) Workshop*, 2013.
- [39] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 807–814, 2010.
- [40] M. S. Osman, H. Chen, K. Creamer, J. Minotto, J. Liu, S. Mirg, J. Christian, X. Bai, S. Agrawal, and S. R. Kothapalli. A novel matching layer design for improving the performance of transparent ultrasound transducers. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, 69(9):2672–2680, 2022.
- [41] Fernando Perez-Cruz. Kullback–leibler divergence estimation of continuous distributions. *IEEE International Symposium on Information Theory*, pages 1666–1670, 2008.
- [42] Steven F. Railsback and Volker Grimm. *Agent-Based and Individual-Based Modeling: A Practical Introduction*. Princeton University Press, 2011.
- [43] Steven F. Railsback and Volker Grimm. *Agent-Based and Individual-Based Modeling: A Practical Introduction*. Princeton University Press, 2012.
- [44] Christian Robert and George Casella. *Monte Carlo Statistical Methods*. Springer, 2004.
- [45] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations (ICLR)*, 2020.
- [46] Murray Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832–837, 1956.

- [47] Timothy J. Rudge, Patrick J. Steiner, Andrew Phillips, and Jim Haseloff. Computational modeling of synthetic microbial biofilms. *ACS Synthetic Biology*, 1(8):345–352, 2012.
- [48] Timothy J. Rudge, Paul J. Steiner, Andrew Phillips, and Jim Haseloff. Cellmodeller: A python framework for modelling growth of bacterial colonies. *PLoS ONE*, 7(11):e49177, 2012.
- [49] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 1986.
- [50] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. A gentle introduction to graph neural networks. *Distill*, 6(9):e33, 2021.
- [51] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.
- [52] Scott A Sisson, Yanan Fan, and Mark A Beaumont. *Handbook of Approximate Bayesian Computation*. Chapman and Hall/CRC, 2018.
- [53] Scott A. Sisson, Yanan Fan, and Mark M. Tanaka. Sequential monte carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 104(6):1760–1765, 2007.
- [54] Samuel L Smith and Quoc V Le. Don’t decay the learning rate, increase the batch size. *ICLR*, 2018.
- [55] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.
- [56] David R. Stirling, Margaret J. Swain-Bowden, Andrew M. Lucas, Anne E. Carpenter, et al. Cellprofiler 4: Improvements in speed, utility, and usability. *BMC Bioinformatics*, 22:1–11, 2021.
- [57] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *International Conference on Learning Representations (ICLR)*, 2018.
- [58] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

- [59] Minjie Wang and Jure Leskovec. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2020.
- [60] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(4):1109–1138, 2021.
- [61] Hideyoshi Yanagisawa, Oto Kawamata, and Kazutaka Ueda. Modeling emotions associated with novelty at variable uncertainty levels: A bayesian approach. *Frontiers in Computational Neuroscience*, 13, 2019. Editor’s Pick 2021.
- [62] A. Yip, J. Smith-Roberge, S. H. Khorasani, M. G. Aucoin, and B. P. Ingalls. Calibrating spatiotemporal models of microbial communities to microscopy data: A review. *PLOS Computational Biology*, 18(10):e1010533, 2022.

# APPENDICES

# Appendix A

## More Details about Data and Features

This section provides a complete description of all features reported by TrackRefiner [1]. Spatial quantities are expressed in micrometers ( $\mu\text{m}$ ) after conversion from pixel units using a user-provided conversion factor. Time-dependent quantities are reported in physical time units or normalized per time step. The orientation feature is converted from degrees to radians according to:

$$\theta = -\frac{(\text{Orientation (degrees)} + 90)\pi}{180}.$$

### Link and Motion Features

- **TrajectoryX**: Displacement in the x-direction between linked bacteria (continuity or division). Set to NaN for unexpected beginnings and bacteria in the first time step.
- **TrajectoryY**: Displacement in the y-direction between linked bacteria (continuity or division). Set to NaN for unexpected beginnings and bacteria in the first time step.
- **Bacterium\_Movement**: Minimum displacement among center and endpoints between source and target. Set to NaN for division links, unexpected beginnings, and first time step.

- **Direction\_of\_Motion:** Angle of motion between linked bacteria. Set to NaN for unexpected beginnings and first time step.
- **Instant\_Velocity:** Frame-to-frame velocity. Set to NaN at the first time step.
- **Average\_Instant\_Velocity:** Mean of Instant\_Velocity over the life history. Set to NaN if life history length is 1.
- **Velocity:** Average velocity over the full life cycle.

## Growth Features

- **Instant\_Elongation\_Rate:** Frame-to-frame elongation rate. Set to NaN at the first time step.
- **Elongation\_Rate:** Average elongation rate over the life cycle.
- **Length\_Change\_Ratio:** Ratio of target length to source length. Set to NaN for division events, unexpected beginnings, first time step, and life history length of 1.
- **Avg\_Length\_Change\_Ratio:** Average of Length\_Change\_Ratio over life history. Set to NaN if life history length is 1.
- **strainRate:** Relative length change between consecutive frames.
- **strainRate\_rolling:** Rolling average of strain rate over time.
- **Average\_Length:** Mean bacterium length over its life history.

## Geometry and Position Features

- **AreaShape\_Center\_X, AreaShape\_Center\_Y:** Center coordinates converted from pixels to micrometers.
- **Location\_Center\_X, Location\_Center\_Y:** Alternative center coordinates converted to micrometers.
- **AreaShape\_MajorAxisLength:** Major axis length (cell length) in micrometers.
- **AreaShape\_MinorAxisLength:** Minor axis length (cell width) in micrometers.

- **radius:** Half of the minor axis length.
- **AreaShape\_Orientation:** Orientation in radians after conversion from degrees.
- **dir:** Unit direction vector defined as  $[\cos(\theta), \sin(\theta)]$ .
- **Endpoint1\_X, Endpoint1\_Y:** Coordinates of the first endpoint.
- **Endpoint2\_X, Endpoint2\_Y:** Coordinates of the second endpoint.
- **Bacterium\_Slope:** Slope of the line defined by endpoints. Set to NaN for vertical lines.
- **Orientation\_Angle\_Between\_Slopes:** Angle between slopes of linked bacteria. Set to NaN for first time step or unexpected beginnings.

## Lineage and Lifecycle Features

- **id:** Unique identifier for each bacterium.
- **parent\_id:** Identifier of the mother bacterium. Set to 0 for initial or unexpected entries.
- **divideFlag:** Indicates whether the bacterium undergoes division.
- **Division\_TimeStep:** Time step at which division occurs. Set to 0 if no division occurs.
- **cellAge:** Age of the bacterium in time steps.
- **LifeHistory:** Total number of time steps the bacterium exists.
- **Unexpected\_Beginning:** True if bacterium appears without a previous match.
- **Unexpected\_End:** True if bacterium disappears before final frame without division.
- **Division\_Family\_Count:** Total number of divisions in the lineage tree.
- **startVol:** Length at birth (first time step).
- **targetVol:** Length at final time step.

## Division-Specific Features

- **Daughter\_Mother\_Length\_Ratio:** Ratio of daughter length to mother length at division.
- **Total\_Daughter\_Mother\_Length\_Ratio:** Total daughter length divided by mother length.
- **Max\_Daughter\_Mother\_Length\_Ratio:** Maximum daughter length divided by mother length.
- **Daughter\_Avg\_TrajectoryX, Daughter\_Avg\_TrajectoryY:** Average displacement of daughters in x and y directions.

## Previous-Step Features

- **Prev\_Bacterium\_Slope:** Slope at previous time step.
- **Prev\_MajorAxisLength:** Length at previous time step.
- **Prev\_Center\_X, Prev\_Center\_Y:** Previous center coordinates.
- **Prev\_Endpoint1\_X, Prev\_Endpoint1\_Y:** Previous endpoint 1 coordinates.
- **Prev\_Endpoint2\_X, Prev\_Endpoint2\_Y:** Previous endpoint 2 coordinates.

## Neighborhood Features

- **Neighbor\_Difference\_Count:** Number of different neighbors between linked bacteria.
- **Neighbor\_Shared\_Count:** Number of shared neighbors.
- **Source\_Neighbor\_Avg\_TrajectoryX, Source\_Neighbor\_Avg\_TrajectoryY:** Average displacement of neighboring bacteria.
- **Motion\_Alignment\_Angle:** Normalized angle between bacterium motion and neighbor motion.

## Other Features

- **time**: Real time obtained from time step and interval.
- **cellType**: Cell classification based on fluorescence intensity across channels.

The following features are specified by the **TrackRefiner** pipeline. For details, see the **TrackRefiner** documentation page by Dr. Atiyeh Ahmadi [1]. Several normalization methods are applied, depending on the feature type.

Table A.1: Feature List with Data Types and Normalization

| Feature Name                       | Data Type          | Normalization |
|------------------------------------|--------------------|---------------|
| ImageNumber                        | (discrete) int     | N/A           |
| ObjectNumber                       | (discrete) int     | N/A           |
| id                                 | (discrete) int     | N/A           |
| parent_id                          | (discrete) int     | N/A           |
| Unnamed: 0                         | (discrete) int     | N/A           |
| ImageName                          | (discrete) int     | N/A           |
| TrajectoryX                        | (continuous) float | standard      |
| TrajectoryY                        | (continuous) float | standard      |
| Endpoint1_X                        | (continuous) float | standard      |
| Endpoint1_Y                        | (continuous) float | standard      |
| Endpoint2_X                        | (continuous) float | standard      |
| Endpoint2_Y                        | (continuous) float | standard      |
| divideFlag                         | (categorical) bool | one-hot       |
| Division_TimeStep                  | (discrete) int     | standard      |
| Daughter_Mother_Length_Ratio       | (continuous) float | standard      |
| Total_Daughter_Mother_Length_Ratio | (continuous) float | standard      |
| Max_Daughter_Mother_Length_Ratio   | (continuous) float | standard      |
| Daughter_Avg_TrajectoryX           | (continuous) float | standard      |
| Daughter_Avg_TrajectoryY           | (continuous) float | standard      |
| Unexpected_End                     | (categorical) bool | one-hot       |
| Unexpected_Beginning               | (categorical) bool | one-hot       |
| LifeHistory                        | (discrete) int     | standard      |
| Bacterium_Movement                 | (continuous) float | standard      |
| Bacterium_Slope                    | (continuous) float | standard      |

| Feature Name                       | Data Type          | Normalization |
|------------------------------------|--------------------|---------------|
| Orientation_Angle_Between_Slopes   | (continuous) float | standard      |
| Length_Change_Ratio                | (continuous) float | standard      |
| Direction_of_Motion                | (continuous) float | standard      |
| Avg_Length_Change_Ratio            | (continuous) float | standard      |
| cellAge                            | (discrete) int     | standard      |
| radius                             | (continuous) float | standard      |
| time                               | (discrete) int     | standard      |
| dir                                | (continuous) float | standard      |
| startVol                           | (continuous) float | standard      |
| targetVol                          | (continuous) float | standard      |
| Neighbor_Difference_Count          | (discrete) int     | standard      |
| Neighbor_Shared_Count              | (discrete) int     | standard      |
| Elongation_Rate                    | (continuous) float | standard      |
| Velocity                           | (continuous) float | standard      |
| strainRate                         | (continuous) float | standard      |
| strainRate_rolling                 | (continuous) float | standard      |
| cellType                           | (discrete) int     | standard      |
| Source_Neighbor_Avg_TrajectoryX    | (continuous) float | standard      |
| Source_Neighbor_Avg_TrajectoryY    | (continuous) float | standard      |
| Motion_Alignment_Angle             | (continuous) float | standard      |
| Average_Length                     | (continuous) float | standard      |
| Division_Family_Count              | (discrete) int     | standard      |
| Instant_Elongation_Rate            | (continuous) float | standard      |
| Instant_Velocity                   | (continuous) float | standard      |
| Average_Instant_Velocity           | (continuous) float | standard      |
| YFP                                | (discrete) int     | standard      |
| TrackObjects_Label_50              | (discrete) int     | standard      |
| TrackObjects_ParentImageNumber_50  | (discrete) int     | standard      |
| TrackObjects_ParentObjectNumber_50 | (discrete) int     | standard      |
| Prev_Endpoint1_X                   | (continuous) float | standard      |
| Prev_Endpoint1_Y                   | (continuous) float | standard      |
| Prev_Endpoint2_X                   | (continuous) float | standard      |
| Prev_Endpoint2_Y                   | (continuous) float | standard      |
| Prev_Center_X                      | (continuous) float | standard      |
| Prev_Center_Y                      | (continuous) float | standard      |
| Prev_MajorAxisLength               | (continuous) float | standard      |

| Feature Name              | Data Type          | Normalization |
|---------------------------|--------------------|---------------|
| Prev_Bacterium_Slope      | (continuous) float | standard      |
| AreaShape_Center_X        | (continuous) float | standard      |
| AreaShape_Center_Y        | (continuous) float | standard      |
| Location_Center_X         | (continuous) float | standard      |
| Location_Center_Y         | (continuous) float | standard      |
| AreaShape_MajorAxisLength | (continuous) float | standard      |
| AreaShape_MinorAxisLength | (continuous) float | standard      |
| AreaShape_Orientation     | (continuous) float | standard      |
| AreaShape_Area            | (continuous) float | standard      |

# Appendix B

## Results of Previous Case Study

As described in [subsection 1.6.6](#), my seniors conducted a case study using 6 conventional summary statistics proposed based on biological expertise [\[2\]](#).

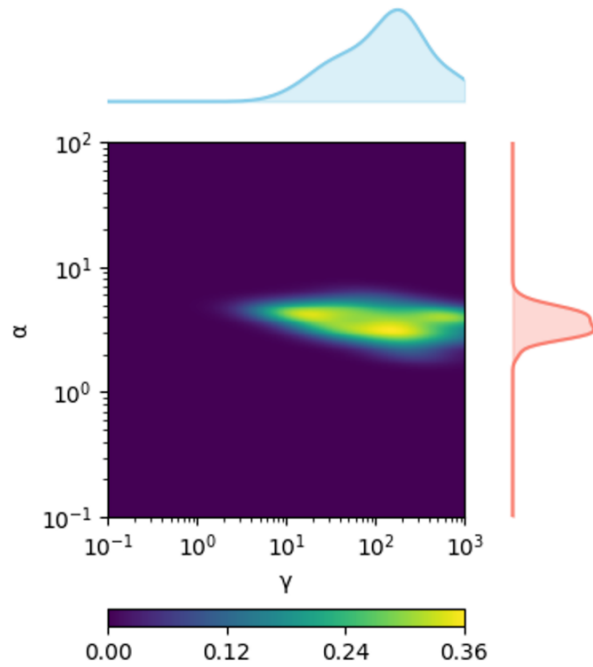


Figure B.1: Posterior probability densities for calibrated parameter values of  $\gamma$  and  $\alpha$  after six rounds of ABC calibration [\[2\]](#).

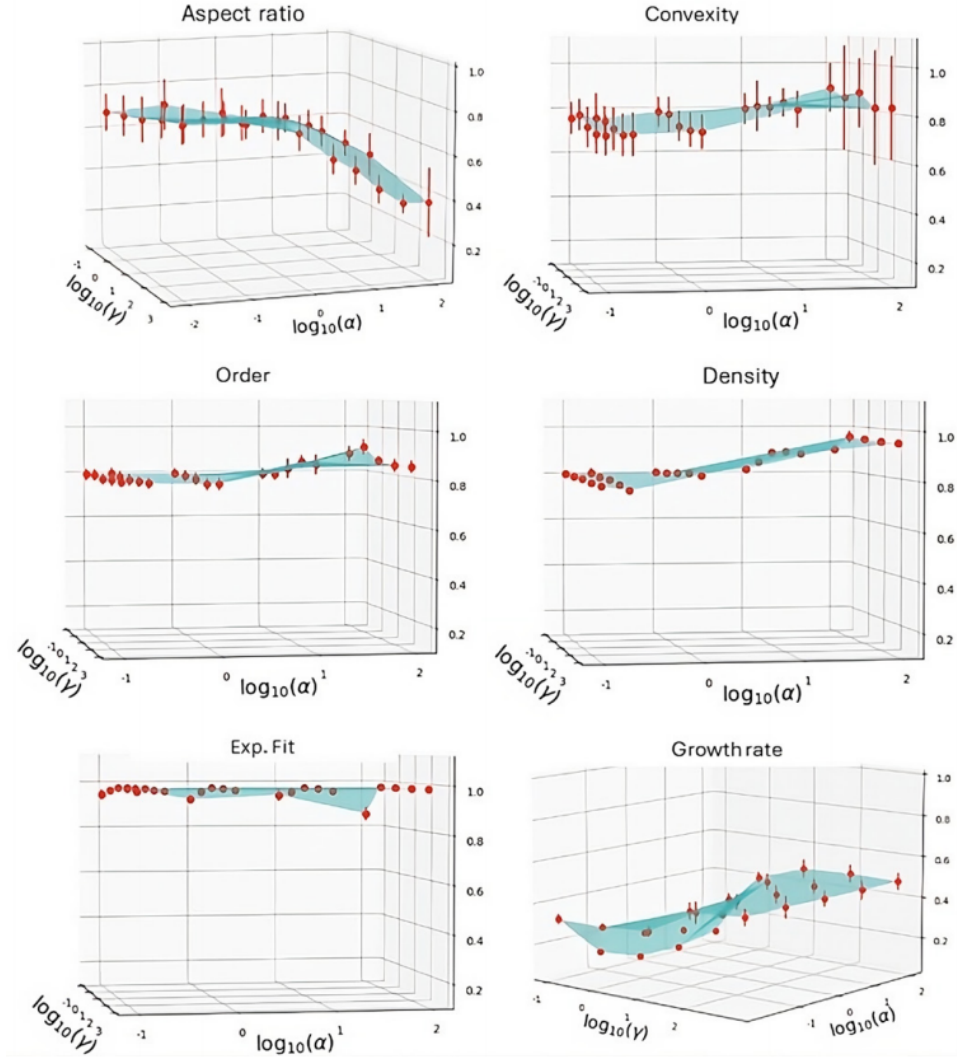


Figure B.2: Global sensitivity analysis across  $\gamma$  and  $\alpha$  values. Red bars represent the standard deviation of each feature value determined from an ensemble of 50 simulations for each parameter pair [2].

Figure B.2 shows the sensitivity analysis result of 6 summary statistics that were proposed manually. Figure B.1 shows the posterior inference result from ABC-SMC pipeline with respect to parameter  $\gamma$  and  $\alpha$ .

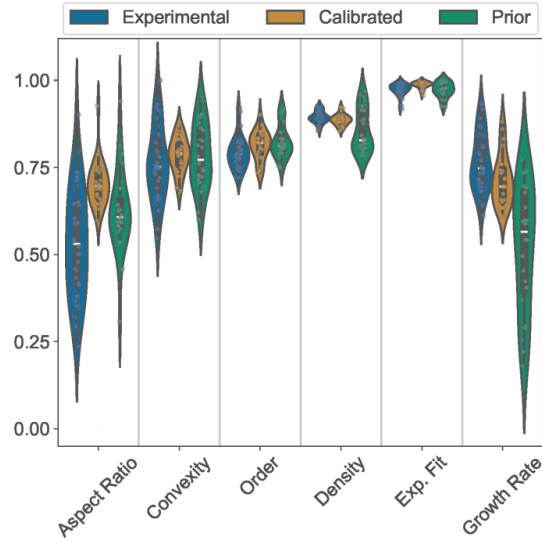


Figure B.3: Summary statistics for experimental data and simulations sampled from prior distributions and final posterior distribution ( $n=24$ ). Lower box boundary: 25th percentile; upper box boundary: 75th percentile; line inside box: median; whiskers:  $1.5 \times$  interquartile range [2].

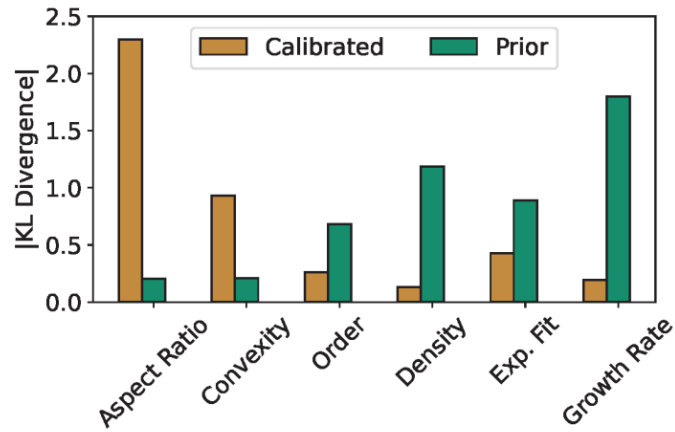


Figure B.4: KL divergence between experimental data and simulations sampled from prior distributions and final posterior distributions shown in Figure B.3 [2].

My seniors randomly took 24 parameter values from the inferred posterior and the

initial prior [2]. Then use these parameter values to simulate and get the 6 summary statistics that formed the distributions as shown in Figure B.3. Afterward, they computed the KL divergence between the posterior and experimental results, and between the prior and experimental results, respectively. Eventually, they got Figure B.4 which clearly shows that the first 2 summary statistics (i.e., **Aspect Ratio** and **Convexity**) performed poorly. Note that Figure 3.21 is the reproduced result that I got with respect to Figure B.3.