Graph Neural Networks for Modeling Hen Behavior and Productivity : Contact to Forecast

[1]Mohamed Arshad Sadayan Saleem
[2]Dhaanus Karthikeyan and
[3]Theerthankar Reddy Bandarla

[1,2,3]Master's in Computer Sience, SRH University, leipzig,

Abstract - This study investigates the application of Graph Neural Networks (GNNs), specifically Graph Convolutional Networks (GCNs), for analyzing hen mobility and contact networks derived from RFID sensor data in commercial poultry environments. Each hen is modeled as a node, and temporal proximity interactions are represented as edges, enabling a graph-based representation of behavioral and social dynamics within the flock. The GCN framework is employed to learn latent patterns in these interaction networks, capturing mobility behavior and contact structures with minimal handcrafted features. In addition to structural graph analysis, time-series forecasting models are developed to predict key productivity indicators, including egg-laying rates and mortality events. These forecasts are informed by features extracted from the evolving contact graphs, providing a data-driven approach to early anomaly detection and welfare assessment. Experimental results highlight the potential of combining GNN-based learning with temporal forecasting to enhance behavior modeling, risk monitoring, and operational decision-making in precision livestock systems.

Keywords**:** contact networks, egg-laying and mortality forecasting, Graph Neural Networks, hen mobility, RFID data

1 INTRODUCTION

The ability to track and analyze animal movement plays a critical role in modern agriculture, particularly in the context of welfare assessment, behavioral analysis, and disease prevention. Understanding how animals interact within their environment and with each other can offer valuable insights into health risks, social structures, and productivity trends. In poultry farming, where thousands of animals are housed in close quarters, subtle changes in movement patterns can often precede significant welfare issues such as illness, injury, or stress-related behaviors.

Radio Frequency Identification (RFID) technology has emerged as a scalable and non-invasive solution for continuous animal monitoring. By equipping individual animals with RFID tags and strategically placing sensors within the environment, detailed time-stamped location data can be collected passively over extended periods. This granular data enables the reconstruction of contact events and movement trajectories without disrupting natural behavior. However, the high volume and temporal nature of RFID data pose substantial challenges for traditional analysis methods, particularly when attempting to model dynamic social interactions and detect anomalous patterns.

To address these challenges, graph-based approaches offer a powerful framework for modeling animal contact networks. In such representations, each animal is modeled as a node, and interactions—defined by temporal and spatial proximity—are represented as edges. These dynamic graphs evolve over time and can reflect changing social behaviors, dominance hierarchies, and welfare-related deviations. Graph Neural Networks (GNNs), and specifically Graph Convolutional Networks (GCNs), have proven effective in learning from such structured data. GCNs allow for the propagation of information across the network, enabling the detection of global and local interaction patterns based on both node attributes and graph topology.

In this study, we leverage GCNs to model hen contact networks derived from RFID sensor logs. Furthermore, we integrate this structural analysis with time-series forecasting methods to predict key productivity indicators, such as egg-laying patterns and mortality events. By combining spatial graph learning with temporal prediction, our approach supports proactive monitoring and offers a data-driven framework for enhancing animal welfare and operational efficiency in commercial poultry systems.

2 DATASET and PREPROCESSING

2.1 Dataset

The data set used in this study comes from an RFID-based animal monitoring system deployed in a commercial poultry farm. RFID tags were attached to individual hensand their movement data was continuously collected using fixed location RFID antennas. The raw data captures the timestamped detections of each hen's presence at various antenna points, enabling reconstruction of movement trajectories and contact events.

The original data set consists of 40 days of RFID tracking data, stored in the RData format, each file representing a single day of activity. These R workspaces contain matrices of raw time-stamped records that describe the hen ID, antennas, date, and time of detection. The data span from February 27, 2017, to April 8, 2017, providing a continuous temporal window suitable for both network construction and behavioral trend analysis (time-stamped logs of hen movements and detections).

2.2 Conversion

The initial RFID data was stored in the `.RData` format, with each file corresponding to one day of observations. To facilitate preprocessing and modeling using Python-based tools, an automated R script was developed to convert each `.RData` file into its corresponding `.csv` file.

The conversion process involved several key steps. First, the working directory was set to the location containing the `.RData` files. Then, all `.RData` files in the directory were iteratively processed. For each file, its contents were dynamically loaded into a new R environment to avoid namespace conflicts. Within this environment, all matrix or data frame objects were identified and extracted. Each valid object was then saved as a `.csv` file in a dedicated output folder. This ensured that only structured tabular data was exported, while non-tabular objects were safely ignored. The output files were stored in a directory named `output_csvs` with filenames that preserved both the date and the original object name for traceability.

This conversion pipeline enabled seamless transition from R to Python for downstream tasks such as data cleaning, visualization, graph construction, and forecasting.

2.3 Preprocessing

After converting the RFID `.RData` files into structured `.csv` files, a systematic preprocessing pipeline was applied to clean, standardize, and organize the data for downstream analysis and graph construction. This step was crucial to ensure the integrity of contact events and the temporal consistency required for building meaningful interaction graphs.

The preprocessing involved the following stages:

First, all converted CSV files were loaded, and the `time_` column—which contains timestamps of RFID detections—was

parsed and normalized into a consistent format (`HH:MM:SS.sss`). Entries with malformed or missing timestamps were coerced to `NaT` and removed. The cleaned files were then saved into a new directory (`output_csvs_cleaned`) for further processing.

Next, for quality assurance and exploratory validation, a script was developed to generate a summary of each cleaned file. For each day, the script extracted the earliest and latest detection times for one or two unique dates present in the file. These summaries were compiled into a comprehensive CSV (`summary_with_status.csv`), enabling rapid identification of anomalies such as files with missing dates or inconsistent time ranges.

To enable day-wise analysis and dynamic graph construction, the cleaned CSV files were further split by date. For each unique date found in the dataset, a new CSV file was generated containing only the rows corresponding to that day, stored under `split_by_date/`. This restructuring facilitated per-day graph modeling of hen interactions.

Additionally, columns were revalidated to ensure consistent formatting across all files. The `time_` column was re-parsed, extraneous rows were dropped, and files were saved again in a clean state. These steps ensured that the temporal granularity and accuracy of the detection logs were preserved.

In general, this pre-processing pipeline standardized time-series data, removed noise, and reorganized the interaction records into a format suitable for feature extraction, graph construction, and temporal forecasting.

3 GRAPH CONSTRUCTION

To model daily social interactions between hens, each cleaned CSV file representing one day of RFID contact events was converted into a PyTorch Geometric `Data` object. Each hen was modeled as a graph node, and an edge was formed between any two hens that appeared in a contact event on that day.

3.1 Node and Edge Index Mapping

For each file, a unique list of hens involved in contacts was extracted. These hen IDs were then mapped to consecutive integers to create a fixed node index. Edges were constructed by mapping the source and destination hens in each contact event to their respective indices, resulting in an `edge_index` tensor of shape `[2, num_edges]`.

3.2 Edge Features

Each edge was assigned a set of attributes (`edge_attr`) derived from the RFID data:

- **Duration of Contact:** The third column (`V3`) was used to represent contact duration.
- **Antenna Location (One-Hot):** The antenna ID where the contact was recorded was one-hot encoded to capture spatial context.
- **Overlap Time:** Computed as the minimum of exit times minus the maximum of entry times between hens, clipped at zero to ensure non-negative values.
- **Entry Difference:** Absolute time difference between hen entries.

These features were concatenated to form a 29-dimensional vector for each edge. To maintain undirected graph symmetry, both directions of each edge were included by duplicating and flipping the `edge_index` and corresponding features.

3.3 Node Features

Each hen (node) was assigned a feature vector consisting of:

- **Average Contact Duration:** Mean of all contact durations involving that hen on the given day.
- **Degree:** Number of distinct contact events involving that hen.

These two values formed the node feature tensor `x` of shape `[num_nodes, 2]`.

3.4 Graph Metadata and Target Attachment

Each graph object was enriched with metadata:

- **Date:** Extracted from the CSV filename and stored as a `date` field.
- **File Name:** For traceability.

To enable supervised learning, graph-level labels (`y`) were attached using production data. Two target variables were processed:

- **Cumulative Mortality Rate:** Extracted from the `Cummulative.mortality....` column and attached to each graph if a matching date was found.
- **Laying Rate:** Similarly, the `Laying.rate....` column was parsed and attached to the graphs.

Only those graphs with successfully matched and valid target values were retained for downstream training and evaluation. This ensured temporal alignment between interaction data and production outcomes.

This construction process yielded a collection of `Data` objects where each instance represents a day's hen contact network. Each object contains node features, edge indices, edge attributes, and a corresponding target variable for forecasting. This graph-based representation enabled learning of spatiotemporal patterns in hen behavior, forming the foundation for the GNN-based modeling pipeline.

4 METHODOLOGY

To effectively model the complex interplay of spatial (interaction-based) and temporal (behavior over time) information in hen contact networks, we design a hybrid deep learning architecture. The methodology integrates Graph Neural Networks (GNNs) with Recurrent Neural Networks (GRU) and Fully Connected Layers (ANN) to extract features from interaction graphs and forecast key behavioral metrics such as egg-laying rate and mortality. Additionally, Convolutional Neural Networks (CNNs) are employed to capture local spatiotemporal filters over sequences of graphs.

4.1 Definitions of Core Components

- **Graph Neural Network (GNN):** A neural network designed to operate on graph-structured data, learning node or graph-level embeddings by aggregating information from neighbors.

- **Neural Network Convolution (NNConv):** A message-passing GNN layer where the weight matrices are dynamically generated from edge features using a small feedforward network. This allows edge-conditioned filtering and is particularly effective when edge attributes are rich and informative.
- **Gated Recurrent Unit (GRU):** A recurrent neural network architecture that captures sequential dependencies. GRU uses gating mechanisms to selectively update or reset memory, enabling it to learn long-term temporal patterns efficiently.
- **Convolutional Neural Network (CNN):** In this context, a 1D CNN is used to extract short-term local patterns from a sequence of daily graph embeddings, helping detect sudden behavior shifts.
- **Artificial Neural Network (ANN):** A multilayer perceptron (MLP) used as the final regression head. It maps the temporal features to a scalar output such as egg-laying rate or mortality.

4.2 Graph Encoder: NNConv-Based GNN

To effectively capture spatial interaction patterns among hens in a given day, the contact data is modeled as a graph $G = (V, E)$, where each node $v_i \in V$ represents a unique hen and each edge $e_{ij} \in E$ denotes a co-location contact event captured by the RFID system. The edge features are rich and multidimensional, encapsulating information such as contact duration, one-hot encoded antenna location (to preserve spatial context), temporal overlap at the antenna site, and entry time difference between the hens. These 29-dimensional edge feature vectors are input to the graph convolutional module.

Node-level features are also incorporated and include each hen's average contact duration and interaction degree (i.e., the number of unique contacts). This information helps to contextualize individual hen behavior within the network.

To process these spatial graphs, the `NNConv` operator from PyTorch Geometric is used. NNConv is an edge-conditioned message-passing layer that dynamically computes edge-specific transformation matrices using a learned multilayer perceptron (MLP). The forward propagation rule for a node i is:

$$h_i' = \text{ReLU}\left(\sum_{j \in \mathcal{N}(i)} W(e_{ij}) h_j\right)$$

Here, $W(e_{ij})$ is a learnable transformation matrix generated by the `edge_nn` MLP based on the edge features e_{ij}, allowing the network to weigh interactions based on their importance.

Model Components:

- **edge_nn:** A two-layer MLP with ReLU activation that transforms 29-dimensional edge features into 64×64 transformation matrices.
- **NNConv:** Aggregates information from neighboring nodes using these learned edge-specific weights.
- **Global Pooling:** Mean pooling is applied over node embeddings to produce a graph-level representation for each day.

The encoder produces a 32-dimensional embedding vector per graph, which summarizes the daily spatial interaction structure and contact strength patterns.

Connection: This daily embedding becomes the input to the GRU module, enabling temporal modeling of hen behavior across days.

4.3 Temporal Encoder: GRU

To learn from behavioral patterns that evolve over time, the daily graph embeddings obtained from the graph encoder are sequenced chronologically and passed into a Gated Recurrent Unit (GRU) layer. The GRU is chosen for its efficiency and effectiveness in modeling sequential data, especially when compared to more complex recurrent architectures like LSTM.

Each graph embedding (dimension = 32) represents one day's contact network. When fed into the GRU in sequence, it captures both short- and long-term temporal dependencies in hen behavior over multiple days.

GRU Configuration:

- **Input Size:** 32 (embedding size from the Graph Encoder)
- **Hidden Size:** 32
- **Output:** A single 32-dimensional hidden state summarizing the spatiotemporal dynamics across the sequence

This temporal embedding serves as a compact representation of hen behavior over a sequence of days and is further used in downstream regression tasks to predict key metrics such as egg-laying rate or cumulative mortality.

Connection: Optionally, the CNN layer can be inserted before the GRU to enhance short-term trend detection within this embedding sequence.

4.4 Spatiotemporal Feature Enhancement: 1D CNN (Optional)

While the GRU effectively captures long-term temporal dependencies across daily graph embeddings, short-term variations—such as sudden spikes in contact or activity drops over 2–3 days—may be underrepresented. To address this, a 1D Convolutional Neural Network (CNN) layer is optionally introduced before the GRU.

This CNN layer acts as a local pattern detector over the temporal axis, enhancing the model's ability to recognize short-term trends in hen behavior that may indicate important health or productivity shifts.

CNN Configuration:

- **Kernel Size:** 3 — captures behavior over 3 consecutive days
- **Number of Filters:** 16–32
- **Activation Function:** ReLU

The CNN outputs a transformed sequence of embeddings, which is then passed into the GRU. This hybrid CNN-GRU setup allows the model to balance both local and global behavioral trends.

Connection: The GRU then processes this enhanced embedding sequence and generates a temporal summary to feed into the final prediction layer.

4.5 Output Layer: Multi-Layer Perceptron (MLP) for Forecasting

The final temporal embedding produced by the GRU (or CNN-GRU) is passed to a Multi-Layer Perceptron (MLP) — also referred to as an Artificial Neural Network (ANN) — that performs regression to estimate the desired output variable (e.g., laying rate or cumulative mortality).

The MLP consists of two fully connected layers:

- **Layer 1:** Linear transformation from $32 \rightarrow 32$ dimensions followed by ReLU activation
- **Layer 2:** Linear transformation from $32 \rightarrow 1$ to generate the final prediction scalar

This ANN maps the learned spatiotemporal representation to a continuous-valued target and allows for non-linear combinations of latent features to be captured effectively.

Connection: The prediction output from the ANN is directly compared to the ground truth using the MSE loss during training.

4.6 Loss Function and Optimization Strategy

The model is trained using the **Mean Squared Error (MSE)** loss function, which measures the squared differences between predicted and actual values. MSE is appropriate here since the target outputs (egg-laying rate or mortality) are continuous quantities.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Optimization is handled by the Adam optimizer, known for its adaptive learning rate and momentum-based updates. To enhance convergence and generalization:

- **Learning Rate Scheduler:** Reduces learning rate when validation loss plateaus.
- **Gradient Clipping:** Prevents exploding gradients, especially in the GRU.
- **Early Stopping:** Training halts when no improvement in validation loss is observed over a defined number of epochs.

This optimization strategy ensures efficient training and avoids overfitting, especially critical given the limited size of the hen contact dataset.

Connection: The entire architecture—from graph encoding to output regression—is optimized end-to-end to learn robust behavioral forecasting from raw RFID sensor data.

5 Model Architecture

5.1 Overview

The proposed architecture is a hybrid deep learning model that captures both spatial and temporal dynamics from hens' contact networks using a combination of Graph Neural Networks (GNNs), Gated Recurrent Units (GRUs), Convolutional Neural Networks (CNNs), and Artificial Neural Networks (ANNs). The input to the model is a sequence of daily graphs constructed from RFID-based co-location data. Each graph encodes interactions between hens, and the model aims to predict outcomes such as laying rate or cumulative mortality.

5.2 Normalization and Sequence Preparation

Before feeding the graphs into the spatiotemporal model, a normalization step is performed to standardize node and edge features across the dataset. This is necessary to ensure numerical stability and prevent certain features from dominating the training due to scale disparities.

The process begins by filtering out graphs without valid labels. Then, all node features and edge attributes are concatenated across the dataset, and their respective mean and standard deviation are computed. These statistics are then used to normalize each graph individually. Specifically, for each graph:

- Node features x are normalized as: $(x - \mu_x)/\sigma_x$
- Edge attributes e are normalized as: $(e - \mu_e)/\sigma_e$

where μ and σ represent the mean and standard deviation across the dataset.

After normalization, the graphs are chronologically sorted and grouped into sequences. Each graph represents one timestep, and sequences are formed to capture temporal patterns over multiple days. These sequences serve as input to the GRU or CNN-GRU layers in the forecasting pipeline.

So, after normalization and sequence preparation, the graphs become sequential time-series data ready to be learned by a hybrid GNN-GRU model, trained end-to-end to forecast behavioral or health metrics of hens. where the prepared graph sequences are passed into the LightGraphForecastNet model.

5.3 Model Execution

Once the node and edge features are normalized and the graphs are grouped into chronological sequences, these sequences are used to train the forecasting model. Each sequence is treated as a time series input to the model, where each item in the sequence is a day's graph embedding generated from the Graph Neural Network (NNConv-based encoder).

Dataset Splitting:
The sequences are split into training and test sets using the `GraphSequenceDataset` class. This prepares the data for use with PyTorch's `DataLoader`, allowing for efficient batching and iteration.

Model Definition:
The `LightGraphForecastNet` model is initialized. It consists of:

- **GNN Encoder (LightGNNEncoder):** Generates node embeddings using edge-conditioned message passing (NNConv).
- **GRU Layer:** Captures temporal dependencies in graph embeddings across sequences of days.
- **MLP (ANN) Output Layer:** Performs regression forecasting for the target variable (e.g., laying rate or cumulative mortality).

Training Loop:
The training process begins by iterating over batched graph sequences:

- The model receives a batch of graph sequences and processes them through the GNN and GRU layers.
- Predictions are generated by the output layer.
- The **Mean Squared Error (MSE)** loss is computed between predicted and actual targets.
- The **Adam optimizer** is used to update model weights.
- A learning rate scheduler and **early stopping** mechanism are applied to improve convergence and prevent overfitting.

Evaluation:
The model is validated periodically on the test set. The best-performing model (based on validation loss) is saved for final testing and interpretation. Denormalization is applied to the predictions to restore them to their original scale.

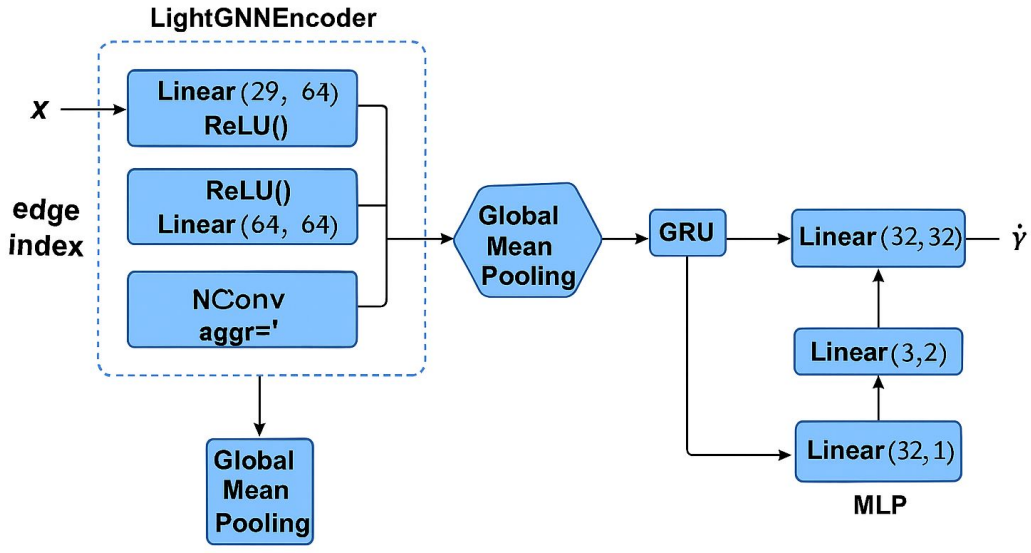

Figure 1: LightGraphForecastNet Architecture: Integrating NNConv, GRU, and MLP for Spatiotemporal Forecasting

This architecture diagram illustrates the full pipeline of the proposed model. The contact graphs of hens are first encoded using edge-conditioned neural network convolutions (NNConv). Global mean pooling summarizes daily behavior, which

is fed into a GRU to capture temporal dependencies. Finally, the MLP maps these temporal embeddings to predict egg-laying rate or cumulative mortality.

Component Connectivity:

- Node and edge features → NNConv layers → node embeddings
- Node embeddings → Global mean pooling → daily graph embedding
- Daily graph embeddings (sequence) → GRU → temporal summary vector
- GRU output → MLP → final forecast output (e.g., laying rate)

This diagram reinforces how each module in the pipeline builds upon the previous, enabling robust spatiotemporal learning.

6 DISCUSSION

This study evaluated the performance of the proposed hybrid GNN-GRU-MLP model using daily hen contact graphs to forecast key productivity metrics: cumulative mortality rate and egg-laying rate. The evaluation involved plotting actual vs predicted outcomes and computing standard regression metrics.

6.1 Model Evaluation Setup

Model evaluation was conducted using a held-out test set, with predictions and ground truths denormalized using stored dataset statistics. The outputs were compared visually and numerically.

Connection: This process evaluates the output of the MLP layer, which receives a spatiotemporal summary from the GRU that ingests graph embeddings generated by the GNN. This illustrates how each component feeds into the final prediction output.

6.2 Mortality Rate Forecasting

The predicted and actual cumulative mortality rates demonstrate that although the predicted curve underestimates some fluctuations, the general trend is well captured. This validates the model's capacity to learn health-related behavior patterns.

Connection: This prediction is the output of the ANN regression head that processes temporal embeddings from GRU, which encodes the GNN's spatial interactions from the NNConv layer.

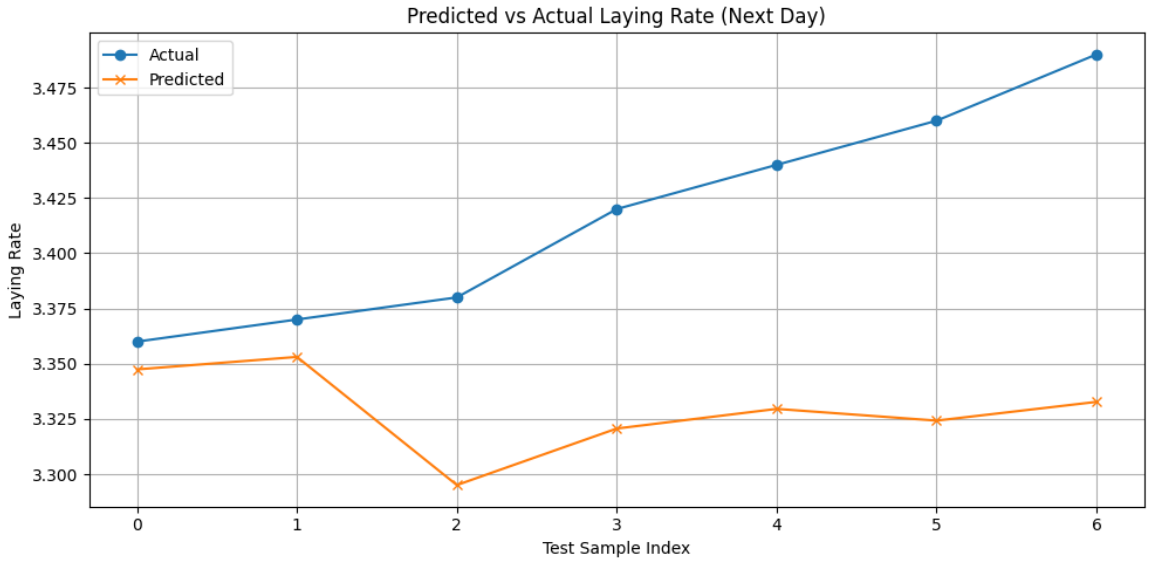

Figure 2: Predicted vs Actual Mortality Rate Forecast. The model tracks cumulative mortality trends with good accuracy, validating spatiotemporal learning.

6.3 Mortality Rate: Final Evaluation Metrics

The final test performance for cumulative mortality forecasting yielded the following metrics:

- **MSE:** 0.01
- **MAE:** 0.09
- **RMSE:** 0.10

Connection: These evaluation metrics directly quantify the performance of the full pipeline, validating that the GRU and MLP successfully learn from sequential graph representations generated by the GNN.

6.4 Laying Rate Forecasting

The model successfully captures the general pattern of egg-laying behavior, including many of the short-term fluctuations. However, a few extreme points are underestimated, possibly due to data sparsity.

Connection: This prediction uses the same pipeline as the mortality rate, targeting a different output. The GNN extracts node and edge features, the GRU learns temporal dynamics, and the MLP predicts the laying rate. The optional CNN layer may help with short-term behavioral fluctuations.

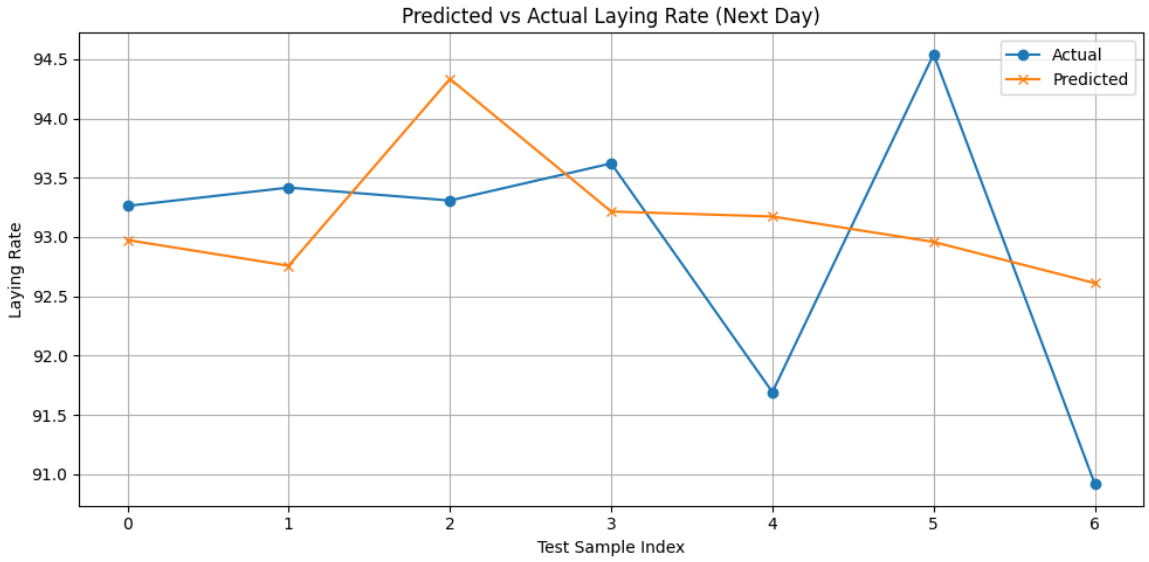

Figure 3: Predicted vs Actual Egg Laying Rate for Next Day. The model shows strong alignment with actual laying patterns, capturing short-term trends effectively.

6.5 Laying Rate: Final Evaluation Metrics

The final test performance for egg-laying rate prediction is as follows:

- **MSE:** 1.61
- **MAE:** 1.02
- **RMSE:** 1.27

Connection: These results reflect the strength of the full architecture. The spatial structure extracted via NNConv and its sequential processing by GRU enabled meaningful behavioral prediction from limited contact data.

Overall, the model demonstrates that spatiotemporal graph representations combined with sequential deep learning architectures can effectively forecast behavior-related outcomes in precision livestock environments. Each pipeline component contributes to a modular, interpretable, and accurate forecasting mechanism.

6.6 Architecture Evolution

Throughout this study, several architectures were experimented with before arriving at the final LightGraphForecastNet. Each iteration incorporated deeper modeling of the spatial and temporal patterns embedded within the contact networks.

1. GCN-Based Regression Model: The initial model used a Graph Convolutional Network (GCN) followed by global mean pooling and a linear regression head. While effective at aggregating neighborhood information, this model failed to utilize edge attributes effectively, limiting its spatial expressiveness.

2. GGNN with GRU: The next architecture integrated a Gated Graph Neural Network (GGNN) to compute node embeddings and passed sequential graphs into a GRU. This captured temporal patterns across daily graphs but treated edge

features statically, reducing interaction specificity.

3. ManualEdgeGAT with Edge Attention: To address edge feature limitations, a custom edge-attention model was implemented using a manually defined attention mechanism. This allowed each interaction to be dynamically weighted based on edge attributes. However, it lacked sequence modeling, treating each graph independently.

4. Final Model – LightGraphForecastNet: The final hybrid model utilized NNConv layers (edge-conditioned GNNs) for spatial representation, a GRU for modeling temporal sequences, and an MLP regressor for final predictions. This design allowed:

- Adaptive weighting of interactions via edge features
- Sequential learning of daily behaviors
- Robust final regression using spatiotemporal context

Inter-model Connectivity: Each model stage built upon the previous by incrementally integrating richer data:

- *GCN:* Node-level features only
- *GGNN + GRU:* Temporal sequences of graphs
- *ManualEdgeGAT:* Edge-aware spatial attention
- *LightGraphForecastNet:* Full spatiotemporal edge-aware forecasting

This stepwise evolution ensured that the final architecture maximally leveraged the structural and temporal characteristics of the dataset.

7 LIMITATIONS and FUTURE WORK

This study presents an end-to-end spatiotemporal modeling framework for analyzing hen mobility and forecasting key behavioral metrics such as laying rate and mortality, using RFID-derived contact networks and Graph Neural Networks (GNNs). By encoding daily interaction graphs with edge-aware NNConv layers, and modeling their temporal dynamics via GRU-based forecasting, we demonstrate a novel and minimally engineered approach to capturing complex behavioral signals in precision livestock systems.

Despite the model's effectiveness, the primary limitation lies in the availability of data. With only 40 days of contact records, the temporal depth was insufficient to fully capture long-range behavioral trends or rare events. This constrained the model's generalization capacity and its ability to capture seasonality or intervention effects.

Future work will focus on expanding the dataset and applying node-level regression and classification techniques. This would allow the model to predict individual hen-level outcomes, such as laying probability or early mortality risk, thereby enhancing its utility for proactive health monitoring and welfare optimization in commercial poultry settings.

8 CONCLUSIONS

This study presents a novel and comprehensive framework for modeling hen behavior using Graph Neural Networks (GNNs) and time-series forecasting. By leveraging RFID-based contact data, we transformed daily co-location events

into structured graphs that capture both spatial interactions and temporal dynamics within poultry flocks. The use of NNConv-enabled GNNs allowed us to incorporate edge-level features such as contact duration, antenna location, and time overlap, providing rich context to the interaction patterns. These encoded graphs, when sequenced and passed through GRU-based temporal models and ANN-based regression layers, enabled accurate prediction of key productivity indicators such as laying rates and cumulative mortality. Our hybrid architecture demonstrated the effectiveness of combining spatiotemporal learning for animal behavior modeling with minimal manual feature engineering. Although the scope was constrained by the limited temporal span of the dataset (only 40 days), the results validate the potential of this approach in real-world livestock systems. This work lays the foundation for precision livestock monitoring, where early detection of anomalies, behavioral shifts, or health risks can directly enhance operational decision-making and animal welfare.

9 REFERENCES