Universiteit
Leiden
The Netherlands

Bachelor Computer Science & Datascience and Artificial Intelligence

Liquid Neural Networks vs RNNs:

A Deep Learning Approach to Time Series Forecasting

Arman Kataria

Supervisor:
Prof. Dr. Michael S. Lew & Dr. Erwin M. Bakker

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

01/07/2025

Abstract

This thesis investigates the performance of Liquid Neural Networks (LNNs) compared to Recurrent Neural Network (RNN) architectures -Vanilla RNN, LSTM, GRU, Phased LSTM, and IndRNN -in time series forecasting tasks across diverse domains, dataset sizes, evaluation metrics, and loss functions. Addressing the main research question, "How do LNNs compare to RNN-based models in time series forecasting?", the study evaluates four sub-questions, focusing on classification and regression metrics, the impact of dataset size, domain-based performance, and the effects of loss functions. Experiments utilize datasets (Ozone, Gesture, Occupancy, Power, and Traffic) and follow established methodologies to ensure robust comparisons. Results demonstrate LNNs' superior adaptability, achieving the highest classification accuracies (e.g., 63.06% on Gesture) and lowest errors (e.g., 0.147 MSE on Traffic), particularly in small-data regimes (2% and 20% Ozone subsets) and with Huber loss (68.40% accuracy on Gesture). GRU excels in regression tasks, while IndRNN shows computational efficiency in large datasets. This study gives a thorough benchmark and offers novel insights into optimizing continuous-time models, thereby open the way for innovation in time series forecasting research and applications.

Contents

1 Introduction

1.1 Background

Time series forecasting is an important component in the decision-making process in various domains such as financial market predictions, ozone layer predictions, and weather predictions, where accurate predictions for future steps are calculated using historical data. The ability to correctly calculate these future steps can affect the strategic planning and resource allocation for a company. For a long time, Traditional statistical methods, such as ARIMA, have been used in dealing with time forecasting due to their high interpretability and theoretical grounding. However, several studies have demonstrated that while interpretable, Traditional statistical methods usually struggle with the data dealing with complex and non-linear relationships. For example, Brockett et al. [1] highlighted the superior flexibility and capabilities of neural networks in tasks related to intricate input-output relationships, even though they may not provide easy interpretability of the process. More recently, Restrepo and Gaviria [2] compared statistical and ANN-based models for demand forecasting and found that neural networks outperformed classical statistical methods, particularly in handling noisy and large datasets. These findings fortify the shift to deep learning networks -especially Recurrent neural networks (RNNs) -for time series forecasting.

As Goodfellow et al. note, "Recurrent neural networks, or RNNs, are a family of neural networks for processing sequential data" [3]. For the time series data, particularly the Recurrent neural networks (RNNs) and their variants, such as Long Short-Term Memory (LSTM) networks[4] and Gated Recurrent Units (GRUs)[5], are widely used due to their capacity to model the temporal data into sequential data. The original form of RNN, often referred to as simple RNN or Vanilla RNN was introduced by Elman [6]. It processes the sequential data via recurrent connections(hidden states that evolve over time), thus enabling the temporal data flow. However, its vulnerability to vanishing and exploding gradients limits the capacity of model long-term dependencies in sequences [7]. To address the problems faced by the Vanilla RNN, Long Short-Term Memory (LSTM) networks [4] introduced gated mechanisms(input, output, and forget gates) to regulate the information flow(from gate to gate), thereby improving the long-term dependencies and retention of memory by using the forget gate mechanisms. The Gated Recurrent Unit (GRU) [5] simplified the LSTM by merging the gates, achieving similar performance with fewer parameters to tune.

The newer innovations in the RNNs include phased LSTM [8], which incorporates the time-dependent gates that allow the network to update selectively based on a periodic function instead of changing hidden states at every time step as done in LSTM. And the Independently Recurrent Neural Network (IndRNN) [9], which uses layer-wise independent recurrent weights that enable the formation of deeper architectures, and also mitigate the issue of gradient vanishing.

Alongside these advancements in the RNN architectures, Liquid Neural Networks (LNNs) proposed by Hasani [10] have emerged as a very new promising approach for time series forecasting. LNNs, unlike traditional RNNs, employ continuous-time computation, thus enabling dynamic adjustment of time constants. This adaptive nature of LNNs allows them to outperform other RNN architectures, especially when the time data is non-stationary.

1.2 Motivation

In this rising age of Artificial Intelligence (AI), the increasing demand for accurate and precise time series forecasting across domains such as finance, energy, and environmental science has driven significant advancements in neural network models and techniques. Traditional RNNs and their variants - including Phased LSTM [8] and IndRNN [9], have become the standard tools for modeling the temporal dependencies in sequential data. Despite their awarding success, these RNN architectures face limitations in handling non-stationary data and dynamically adapting to evolving temporal patterns, especially in complex real-dimensional settings.

The advent of Liquid Neural Networks (LNNs), introduced by Hasani et al. [10], presents a promising alternative to work in real-dimensional settings. Unlike conventional RNNs, LNNs, with the use of dynamic time constants in their continuous-time models, enable them to adjust to non-linear and evolving temporal patterns. Early experiments, as done by Hasani et al. [10], have shown that LNNs exhibit superior robustness and adaptability compared to traditional RNNs in tasks involving non-stationary signals. However, a detailed comparison between LNNs and a thorough set of RNN-based architecture across multiple time-forecasting domains remains limited in current literature.

Moreover, as noted by Bornschein et al. [11], model selection becomes increasingly critical in the small-data regime, where issues of overfitting and underfitting pose significant challenges. This raises questions about how LNNs and RNNs perform under varying dataset sizes, a topic that has received insufficient attention to date. Gütter et al. [12] further underlined how the training set size significantly influences the ability of neural network models to handle data irregularities, such as noise or missing data, introducing the need to evaluate models across different volumes of data.

In addition, most exciting studies evaluate neural networks either on regression-based metrics [13], such as Mean Squared Error (MSE) or Mean Absolute Error (MAE), or classification-based metrics [14], such as precision, recall, and F1-score. Since both types of analysis can reflect different capabilities, it is essential to expand the evaluation scope.

Finally, the role of loss functions is also crucial: recent surveys [15, 16, 17] demonstrate that the choice of loss functions significantly affects both performance and convergence, yet this scope remains underexplored for LNNs.

1.3 Research Questions

Main Question:
How do Liquid Neural Networks compare to RNN-based Neural Networks (Vanilla RNN[6], LSTM[4], GRU[5], Phased LSTM[8] and IndRNN[9]) in time series forecasting tasks?

Sub Questions:

- How do Liquid Neural Networks compare to RNN-based Neural Networks when evaluated using classification and regression metrics?

- How do Liquid Neural Networks and RNN-based Neural Networks perform when trained on datasets of different sizes (small vs. large datasets)?
- How do different models perform across various time series forecasting domains (e.g., financial markets, weather, energy consumption)?
- What impact does the choice of loss function have on the performance of Liquid Neural Networks compared to RNN-based Neural Networks?

These four sub-research questions will guide us to answer the main research question, aiming to systematically assess the effectiveness and flexibility of LNNs to their relative RNNs counterparts. This also provides the basis for a detailed framework to provide a fair and informative comparison.

1.4 Objectives

The objective of this research is to compare Liquid Neural Networks (LNNs) with systematically established and newer Recurrent Neural Network (RNN) architectures in time series forecasting tasks, addressing the research questions through a structured and thorough evaluation.

The study focuses on the following goals:

1. Develop a robust benchmark to assess the performance of LNNs against RNN-based models(Vanilla RNN[6], LSTM[4], GRU[5], Phased LSTM[8] and IndRNN[9]) across time series data, using both regression and classification metrics.
2. Examine the impact of dataset size on the predictive accuracy and robustness of LNNs and RNN-based models, comparing their performance in small-data and large-data regimes to identify scalability and generalization capabilities.
3. Analyze the effect of different loss functions on model training and predictive accuracy, highlighting how different loss functions can impact the model performance across RNNs architectures and identifying optimal loss function strategies for each architecture.
4. Assessing model effectiveness across multiple forecasting domains(e.g., financial markets, weather, energy consumption) to understand cross-domain applicability and adaptability.

By meeting these objectives, this thesis aims to offer practical insights and a rigorous benchmark for evaluating the strengths and limitations of Liquid Neural Networks in comparison to RNN-based models by conducting a comparative analysis using time series forecasting tasks across different domains, data sizes, evaluation metrics, and loss functions.

2 Related Work

2.1 Overview of RNN-based Neural Networks

Feedforward neural networks (FNNs) are the foundational building blocks of deep learning models. FNNs assume fixed-size inputs and outputs, where the information transfer is one-directional -from

input to output -without any feedback loops or cycles[3]. While FNNs have been successfully applied to many problems, they fail to address real-world tasks such as language modeling, speech recognition, and time series forecasting due to their inherent inability to capture temporal dependencies or model sequential data, as each input is treated independently without any notion of order.

Recurrent Neural Networks (RNNs) were developed to address these challenges by incorporating temporal variables in the learning process, allowing them to capture temporal dependencies or model sequential data more naturally. According to Goodfellow et al. [3], RNNs provide a flexible way to share parameters across time steps, which not only reduces model complexity but also enables them to process sequences of varying lengths. This parameter sharing across time steps gives RNNs the ability to generalize patterns over time -an important feature in time series forecasting. However, this flexibility comes with various challenges, such as the network forgetting important information (vanishing gradients) or becoming numerically unstable (exploding gradients).

2.1.1 Vanilla RNN

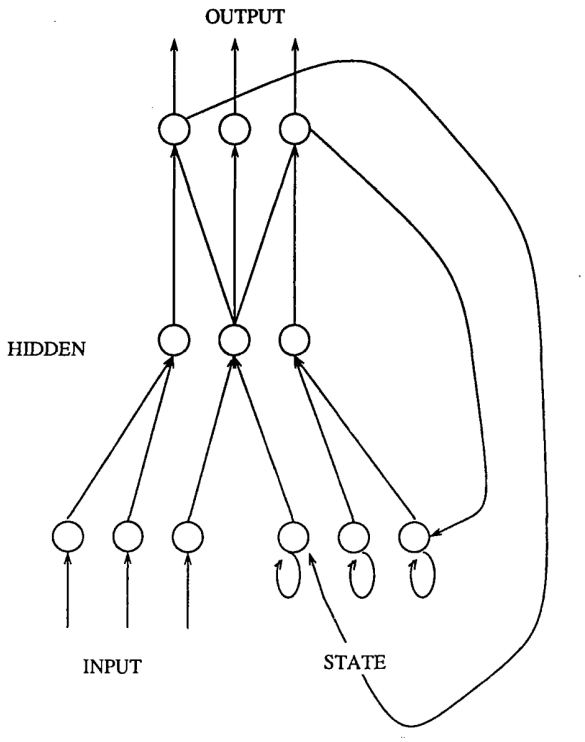

Figure 1: Visual representation of the vanilla RNN structure introduced by Elman[6]

The foundational work on RNNs was introduced by Elman (1990) in his seminal paper, "Finding Structure in Time"[6]. Elman proposed the Simple Recurrent Network (SRN), also known as the Elman network, which utilizes recurrent network connections to equip the network with a form of dynamic memory. In this network, hidden unit patterns are fed back to themselves, allowing the internal structure to reflect the task demands in the context of prior internal states. Figure 1 demonstrates the visualization of the original structure as Elman[6] introduced it. This structure

then enables the network to learn from its past inputs and incorporate that knowledge into its current state, making it suitable for tasks like time series forecasting and natural language processing.

The SRN was demonstrated by simulating simple problems, such as the temporal version of XOR, to discover syntactic and semantic features for words. However, SRN faced challenges in training, particularly with the long sequences, due to the problems of Vanishing and exploding gradients, which limit their ability to capture long-term dependencies effectively [7].

2.1.2 Long Short-Term Memory (LSTM)

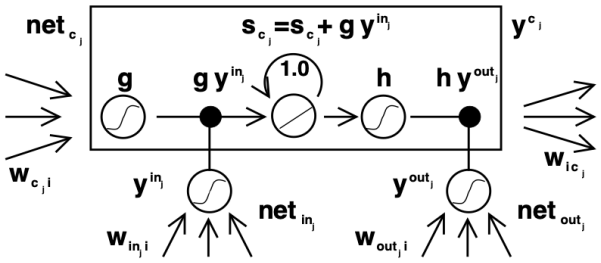

Figure 2: Visual representation of the LSTM structure introduced by Hochreiter and Schmidhuber[4]

To address the vanishing gradient problem in traditional RNNs, Hochreiter and Schmidhuber(1997) introduced Long Short-Term Memory (LSTM) networks in their paper "Long Short-Term Memory"[4]. LSTMs incorporate memory cells and introduce a gated mechanism -input, output, and forget gates -to control and regulate the flow of information. The input gate controls what new information to store, the output gate controls which information to output, and the forget gate controls the information to discard from the memory cell. This gated mechanism enables LSTMs to reduce time lags and retain information over extended time intervals. Figure 2 demonstrates the visualization of the original structure as Hochreiter and Schmidhuber[4] introduced it.

The LSTMs were demonstrated by handling long-term dependencies in artificial data involving local, distributed, real-valued, and noisy pattern representations. LSTMs with a time complexity of O(1) per time step and weight demonstrate that they are computationally efficient, making them a robust choice for tasks like time series forecasting, where long-term patterns are critical.

2.1.3 Gated Recurrent Unit (GRU)

To further reduce the computational complexity while still maintaining to capture the long-term dependencies, Cho et al.(2014) proposed "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation"[5]. Gated Recurrent Units(GRUs) are a simplified variant of LSTMs that use two gates -an update gate and a reset gate -to regulate and control the flow of information. The update gate controls how much of the previous hidden state to retain, while the reset gate controls how the new input is combined with the previous hidden state. Figure 3 demonstrates the visualization of the original structure as Cho et al.[5] introduced it. Unlike LSTMs, GRUs lack a separate memory cell and an additional output gate, resulting in fewer parameters to

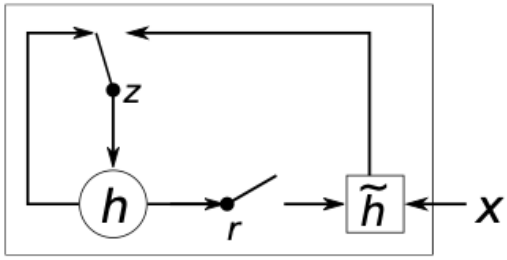

Figure 3: Visual representation of the GRU structure introduced by Cho et al.[5]

train and achieving faster training time.

The GRUs were demonstrated by showing how they improved the performance of statistical machine translation systems by learning semantically and syntactically meaningful representations of linguistic phrases.

2.1.4 Phased LSTM

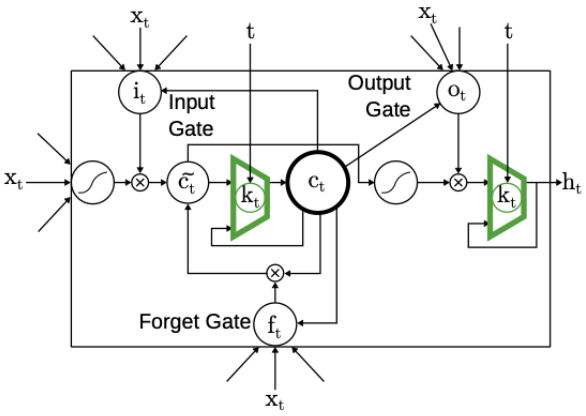

Figure 4: Visual representation of the Phased LSTM structure introduced by Neil et al.[8]

Neil et al.(2016) introduced Phased LSTM in their paper "Phased LSTM: Accelerating Recurrent Network Training for Long or Event-based Sequences" [8]. Phased LSTM extends the LSTM architecture by adding a time gate controlled by a parametrized oscillation. The time gate functions to allow the memory cell to update only during specific phases of the oscillation cycle. Figure 4 illustrates the visualization of the original structure as described by Neil et al.[8] introduced it. This updating mechanism makes Phased LSTM highly efficient for processing irregularly sampled data, such as those from event-driven sensors or multiple sensors with different update intervals.

The Phased LSTMs were demonstrated by showing that they achieved faster convergence than regular LSTMs on tasks requiring the learning of long sequences, even with limited updates, making them well-suited for time series forecasting with irregular sampling.

2.1.5 Independently Recurrent Neural Network (IndRNN)

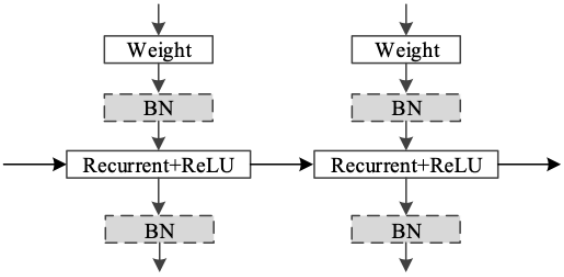

Figure 5: Visual representation of the IndRNN structure introduced by Li et al.[9]

Li et al.(2018) proposed the Independently Recurrent Neural Network (IndRNN) in their paper "Independently Recurrent Neural Network (IndRNN): Building A Longer and Deeper RNN" [9]. In IndRNN, each neuron is independent of the other in the same layer and uses the Hadamard product (element-wise multiplication) for recurrent connections. This structure allows IndRNN to construct longer and deeper networks while mitigating the limitations of traditional RNNs, including gradient vanishing and exploding problems. Figure 5 demonstrates the visualization of the original structure as Li et al.[9] introduced it. Additionally, IndRNN can also work with non-saturated activation functions, making it suitable for tasks that require deep architectures, such as time series forecasting and video analysis.

These advanced RNN architectures have become standard tools for time series forecasting due to their ability to model temporal dependencies more effectively than traditional RNNs. However, they still face limitations, particularly in dealing with non-stationary data where statistical properties change over time and adapting to ever-evolving temporal patterns becomes difficult. This has led to the exploration of other alternative approaches, such as Liquid Neural Networks (LNNs), which will be discussed in the next Section 2.2.

2.2 Liquid Neural Networks

Liquid Neural Networks (LNNs) represent a novel approach to processing sequential data, particularly suited for time series forecasting tasks involving non-stationary and dynamic patterns, introduced by Hasani et al. (2021) in their paper "Liquid Time-constant Networks"[10]. LNNs are inspired by biological neural systems such as those found in the nervous system of the nematode *C. elegans*, created a continuous-time computational framework. LNNs use differential equations to model neuronal dynamics, allowing for time constants to evolve based on the input data, unlike the traditional Recurrent Neural Networks (RNNs).

The key innovation of LNNs lies in their liquid time-constant mechanism, which enables the network to respond to changing input patterns by dynamically adjusting its temporal dynamics. This adaptability makes LNNs effective for non-stationary time series data, where inputs vary over time,

a challenge that traditional RNNs, such as Vanilla RNN [6], LSTM [4], GRU [5], Phased LSTM [8], and IndRNN [9], often struggle to address. Hasani et al.[10] demonstrated that LNNs outperform traditional RNNs in tasks involving non-stationary signals, such as autonomous vehicle trajectory prediction and time series forecasting.

Recent advancements in LNNs have further highlighted their potential. Rubanova et al.(2024)[18] proposed an uncertainty-aware LNN model in their paper "A Novel Uncertainty-Aware Liquid Neural Network for Noise-Resilient Time Series Forecasting and Classification." This model incorporates uncertainty quantification to enhance robustness against noisy data.

LNNs' continuous-time nature requires specialized numerical solvers, which can pose implementation challenges. However, despite this, the adaptability and robustness of LNNs make them a promising alternative for time series forecasting, warranting a detailed comparison with RNN-based architectures, as explored in this thesis later.

2.3 Comparative Studies on RNN Neural Networks for Time Series Forecasting

The application of Recurrent Neural Networks (RNNs) and their variants to time series forecasting has been extensively studied, with numerous research papers comparing their performance across different domains, datasets, and evaluation metrics. This section reviews key comparative studies relevant to the thesis's objective of benchmarking Liquid Neural Networks (LNNs) against RNN-based architectures, including Vanilla RNN[6], LSTM[4], GRU[5], Phased LSTM[8], and IndRNN[9], focusing on their effectiveness in time series forecasting tasks.

Zhou et al.(2023)[19] conducted a comparative analysis of LSTM models integrated with supply chain factors for stock price forecasting. Their study compared LSTM-based models with traditional statistical methods, demonstrating that LSTMs outperform traditional statistical methods in dealing with non-linear patterns in financial time series data. However, this study did not include other RNN models, such as GRUs or IndRNNs, which limits the scope of comparison.

Baihaqi et al.(2023)[14] provided a comprehensive comparison of Bi-LSTM, LSTM, and GRU models for stock price prediction. Their study found that Bi-LSTMs, which process sequences in both forward and backward directions, often achieve higher accuracy than LSTMs and GRUs in stock price prediction. However, GRUs were found to be more computationally efficient, aligning with the findings of Cho et al.(2014) [5]. This study also highlighted the importance of hyperparameter tuning, a factor further explored by Hoque and Aljamaan(2021)[20], who investigated the impact of hyperparameter optimization on machine learning models for stock price forecasting.

Magnusson et al.(2023)[13] explored the use of RNNs, specifically LSTMs, for oil well event prediction. Their study found that LSTMs excel in capturing long-term dependencies in event-driven time series data. However, the study also noted that LSTMs struggle with small datasets, which resonates with the findings of Bornschein et al.(2020)[11] on model selection in the small-data regime. This underscores the importance of evaluating model performance across varying dataset

sizes, as addressed in Sub-RQ2 of this thesis.

A broader perspective is provided by authors from Intel and Ulster University(2022)[21], who conducted a comparative analysis of state-of-the-art time series forecasting algorithms, including RNN-based models. Their study compared LSTMs, GRUs, and other deep learning models across multiple domains, such as finance, energy, and weather forecasting. Their results showed that RNNs performed better in financial forecasting than in weather prediction. These results support the thesis's focus on domain-based comparisons (Sub-RQ3).

Recent research papers have also explored the use of Graph Neural Networks (GNNs) for time series forecasting, which, although not directly related to RNNs, provide insight into advanced neural architectures. Kumbure et al.(2024)[22] researched GNN-based methods for stock market forecasting, concluding that GNNs can model spatial-temporal relationships in financial data, potentially complementing RNNs. Similarly, Jin et al.(2024)[23] conducted research on GNNs for time series tasks, including forecasting and classification.

While these studies provide valuable insights into RNN performance, there is limited research that includes LNNs and their analysis on time forecasting tasks. The limited literature on LNNs, as noted by Hasani et al.(2021)[10] and Rubanova et al.(2024)[18], indicates a gap in comprehensive benchmarks that include both LNNs and a wide range of RNN architectures across multiple domains, dataset sizes, and evaluation metrics. This gap motivates the present thesis, which aims to address these deficiencies by systematically comparing LNNs with RNN-based models and providing a robust benchmark.

3 Methodology

3.1 Models

This section outlines the neural network models employed in the comparative analysis of time series forecasting tasks, addressing the main research question of evaluating Liquid Neural Networks (LNNs) against Recurrent Neural Network (RNN)-based architectures. The models used are Vanilla RNN[6], Long Short-Term Memory (LSTM)[4], Gated Recurrent Unit (GRU)[5], Phased LSTM[8], Independently Recurrent Neural Network (IndRNN)[9], and LNNs[10]. Detailed architectural descriptions are provided in Sections 2.1 and 2.2.

Vanilla RNN. The Vanilla RNN model is implemented using PyTorch's inbuilt `torch.nn.RNN` module, which provides a standard implementation of the Simple Recurrent Network as introduced by Elman(1990)[6]. The model processes sequential inputs through a single recurrent layer, followed by a linear output layer.

Long Short-Term Memory (LSTM). The LSTM model utilizes PyTorch's inbuilt `torch.nn.LSTM` module, based on the architecture proposed by Hochreiter and Schmidhuber(1997)[4]. It employs a single LSTM layer with gated mechanisms, followed by a linear output layer.

Gated Recurrent Unit (GRU). The GRU model is implemented using PyTorch's inbuilt `torch.nn.GRU` module, following the design by Cho et al.(2014)[5]. It consists of a single GRU layer with update and reset gates, followed by a linear output layer.

Phased LSTM. The Phased LSTM model adopts the original implementation, as provided in the accompanying Github code by Neil et al.(2016)[8]. This PyTorch-based implementation includes a time gate controlled by parameterized oscillation, with no modifications made to the original code.

Independently Recurrent Neural Network (IndRNN). The IndRNN model uses the original implementation, as provided in the accompanying Github code by Li et al.(2018)[9]. This PyTorch-based implementation employs independent recurrent weights with element-wise multiplication, with no modifications. The model consists of a single IndRNN layer, preceded by a fully connected layer to process inputs and followed by a linear output layer.

Liquid Neural Network (LNN). The LNN model uses the original implementation, as provided in the accompanying Github code by Hasani et al.(2021)[10]. This TensorFlow 1.15-based implementation models continuous-time dynamics via differential equations, with no modifications made to the original code. The model uses numerical solvers to handle the liquid time-constant mechanism with a linear output layer.

The use of PyTorch-inbuilt models for Vanilla RNN, LSTM, and GRU ensures standardized implementations, facilitating reproducibility. The original author implementations for Phased LSTM, IndRNN, and LNN preserve the integrity of their novel architectures, enabling a fair comparison and reproducibility.

3.2 Datasets and Data Preprocessing

This section describes the datasets and data preprocessing used to evaluate the performance of Liquid Neural Networks (LNNs) and RNN-based models (Vanilla RNN, LSTM, GRU, Phased LSTM, and IndRNN) in time series forecasting tasks, addressing the research questions outlined in Section 1.3. The datasets -Ozone, Gesture, Occupancy, Power, and Traffic -are sourced from the original LNN implementation by Hasani et al.(2021)[10]. These datasets span multiple domains (environmental, human activity, energy, and transportation), supporting Sub-RQ3 on domain-based comparisons. Detailed dataset specifications, including temporal resolution and sources and data preprocessing, are provided in the S5 of the Supplementary Materials in the original paper by Hasani et al.(2021)[10].

The selection of these datasets ensures a diverse evaluation framework covering multiple domains and data characteristics. Timestamps are preserved for Phased LSTM and LNN models, which support event-driven inputs. To address Sub-RQ2, smaller subsets of the ozone data (2%, 20%, and 60% of the entire dataset) are created to simulate small-data regimes, enabling the evaluation of model performance under varying dataset sizes [11].

3.3 Training Configurations

This section details the training configurations for the comparative analysis of Liquid Neural Networks (LNNs) and RNN-based models (Vanilla RNN, LSTM, GRU, Phased LSTM, and IndRNN) on time series forecasting tasks using the Ozone, Gesture, Occupancy, Power, and Traffic datasets. The configurations address the research questions in Section 1.3, ensuring a fair and reproducible evaluation across models, domains (Sub-RQ3), dataset sizes (Sub-RQ2), metrics (Sub-RQ1), and loss functions (Sub-RQ4).

All models are trained without hyperparameter tuning, using default parameters to maintain consistency and simplicity. Each model is trained for 200 epochs with 32 hidden units and optimized using the Adam optimizer with its default settings. The training process involves 10 runs per experiment to account for variability. In these 10 runs, 5 out of 10 results were similar, with 2-3 extreme outliers observed, and the results summarized in respective sub-question Tables are derived from the median observation to ensure robust representation of performance. To account for training variability, each experiment is run 10 times, and performance metrics are measured after removing the extreme outliers (results of 100% or 0%).

The RNN-based models (Vanilla RNN, LSTM, GRU, Phased LSTM, and IndRNN) are implemented in PyTorch and trained using PyCharm, which leverages its integrated development environment for debugging and execution. The LNN model, implemented in TensorFlow 1.15 as per the original author's code by Hasani et al.(2021)[11], is trained using Google Colab, which provides access to GPU resources for the computationally intensive numerical solvers required for continuous-time dynamics.

The training configurations are applied uniformly across the Ozone, Gesture, Occupancy, Power, and Traffic datasets, with smaller subsets (2%, 20% and 60% of the Ozone dataset) used to evaluate performance in small-data and big-data regimes, addressing Sub-RQ2 [12]. The use of default parameters and repeated experiments ensures a fair comparison, minimizing the impact of hyperparameter variations.

3.4 Evaluation Metrics

This section describes the evaluation metrics used to assess the performance of Liquid Neural Networks (LNNs) and RNN-based models (Vanilla RNN, LSTM, GRU, Phased LSTM, and IndRNN) in time series forecasting tasks across the Ozone dataset. The metrics address Sub-RQ1 by enabling a comparison of model performance using both classification and regression metrics while also supporting Sub-RQ2 (dataset size impacts). In Sub-RQ3, selected metrics are used to evaluate the results across different datasets, following a similar approach by Hasani et al.(2021)[10]. Classification metrics include Categorical Cross-Entropy (CCE), Accuracy, Precision, Recall, and F1 Measure. Regression metrics comprise Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Mean Squared Error (MSE). All metrics follow the same experimental setup as described in Section 3.3.

Classification Metrics. For classification tasks, the following metrics are employed:

- **Categorical Cross-Entropy (CCE)** quantifies the divergence between predicted probability distributions and true class labels, serving as both a loss function and a performance metric to evaluate model confidence [3]. Lower CCE values indicate better classification performance.

- **Accuracy** measures the proportion of correctly classified instances, providing a straightforward assessment of overall model performance [16].

- **Precision** calculates the ratio of true positive predictions to total positive predictions, emphasizing the quality of positive class predictions [16].

- **Recall** (or sensitivity) measures the ratio of true positive predictions to actual positive instances, assessing the model's ability to identify all relevant instances [16].

- **F1 Measure** is the harmonic mean of Precision and Recall, providing a balanced metric for classification performance, especially in imbalanced scenarios [16].

Regression Metrics. For regression tasks, the following metrics are used:

- **Root Mean Squared Error (RMSE)** measures the square root of the average squared differences between predicted and actual values, emphasizing larger errors [15].

- **Mean Absolute Error (MAE)** calculates the average absolute differences between predicted and actual values, providing a robust measure of error magnitude that is less sensitive to outliers [15].

- **Mean Squared Error (MSE)** computes the average squared differences between predicted and actual values, serving as a standard metric for regression performance [15].

3.5 Loss Functions

This section outlines the loss functions used to train Liquid Neural Networks (LNNs) and RNN-based models (Vanilla RNN, LSTM, GRU, Phased LSTM, and IndRNN) for time series forecasting tasks on the gesture dataset. The loss functions address Sub-RQ4 by evaluating their impact on model performance through the use of different loss functions during training. The loss functions include MAE, MSE, Huber, and log-cosh loss functions. MSE and MAE are both discussed in section 3.4.
The other two are defined as such:

- **Huber Loss** combines the benefits of MSE and MAE by using a quadratic penalty for small errors and a linear penalty for large errors, controlled by a delta parameter (set to 1.0 by default) [15].

- **Log-Cosh Loss** approximates MSE for small errors and MAE for large errors by computing the logarithm of the hyperbolic cosine of the prediction error [15].

4 Experiments

The experiments are designed to address the main research question: How do Liquid Neural Networks compare to RNN-based Neural Networks in time series forecasting tasks? The evaluation is structured around the four sub-research questions (Sub-RQ1 to Sub-RQ4) outlined in Section 1.3, focusing on metric types, the impact of dataset size, domain-based performance, and the effect of loss functions. The experiments utilize the datasets (Ozone, Gesture, Occupancy, Power, and Traffic) and configurations described in Sections 3.2 and 3.3, with performance assessed using the metrics and loss functions detailed in Sections 3.4 and 3.5.

4.1 Evaluation by Metric Type (Sub-RQ1)

Metrics	Vanilla[6]	LSTM[4]	GRU[5]	Phased LSTM[8]	IndRNN[9]	LNN[10]
RSME	0.50	0.47	0.42	0.48	0.51	0.46
MSE	0.25	0.22	0.18	0.23	0.26	0.21
MAE	0.48	0.38	0.35	0.47	0.50	0.44

	Validation Accuracy	Validation CCE	Test Accuracy	Test CCE	Precision	Recall	F-measure
Vanilla[6]	0.56	0.12	0.58	0.14	0.11	0.76	0.19
LSTM[4]	0.60	0.12	0.60	0.12	0.12	0.86	0.21
GRU[5]	0.68	0.12	0.69	0.13	0.14	0.76	0.24
Phased LSTM[8]	0.61	0.12	0.62	0.13	0.10	0.63	0.17
IndRNN[9]	0.57	0.13	0.59	0.14	0.09	0.61	0.16
LNN[10]	0.63	0.11	0.63	0.12	0.12	0.74	0.20

Table 1: Evaluation Metrics Across Models (ozone dataset)

To address Sub-RQ1 (How do Liquid Neural Networks compare to RNN-based Neural Networks when evaluated using classification and regression metrics?), the performance of all models was evaluated on the Ozone dataset using both regression and classification metrics, as described in Section 3.4. The experimental procedure closely follows the regression-based evaluation framework proposed by Magnusson et al.[13] for assessing recurrent neural networks in time series tasks and the classification-based evaluation approach outlined by Baihaqi et al.[14] for comparing deep learning models in predictive tasks. The results are summarized in Table 1, which presents the performance of each model across these metrics.

The results indicate that GRU outperforms other models in regression tasks, achieving the lowest RMSE (0.42), MSE (0.18), and MAE (0.35), suggesting superior predictive accuracy. LSTM follows closely, with an RMSE of 0.47, MSE of 0.22, and MAE of 0.38, while LNN performs competitively with an RMSE of 0.46, MSE of 0.21, and MAE of 0.44, surpassing Vanilla RNN, Phased LSTM,

and IndRNN. For classification tasks, GRU achieves the highest validation and test accuracies (0.68 and 0.69, respectively) and the highest F-measure (0.24). LSTM achieves the highest recall (0.86), showing its effectiveness in identifying relevant instances. At the same time, LNN performs well with validation and test accuracies of 0.63, the lowest validation CCE (0.11), and a competitive F-measure (0.20). Vanilla RNN and IndRNN exhibit the lowest performance across most metrics, particularly in precision (0.11 and 0.09, respectively) and F-measure (0.19 and 0.16). These findings suggest that GRU and LNN are robust across both regression and classification tasks, with LNN showing particular strength in achieving low CCE, aligning with its adaptability to non-stationary data, as noted by Hasani et al.[10].

4.2 Dataset Size Impact (Sub-RQ2)

	Validation Accuracy	Validation CCE	Test Accuracy	Test CCE	Precision	Recall	F-measure
			Very small 2%				
Vanilla[6]	0.20	0.23	0.15	0.16	0.11	1.00	0.19
LSTM[4]	0.13	0.14	0.30	0.21	0.22	1.00	0.36
GRU[5]	0.27	0.19	0.15	0.20	0.15	1.00	0.26
Phased LSTM[8]	0.40	0.07	0.45	0.11	0.08	1.00	0.15
IndRNN[9]	0.27	0.17	0.25	0.18	0.17	1.00	0.29
LNN[10]	0.60	0.12	0.65	0.12	0.12	1.00	0.22
			small 20%				
Vanilla[6]	0.53	0.14	0.51	0.14	0.13	0.78	0.22
LSTM[4]	0.58	0.13	0.58	0.12	0.12	0.79	0.21
GRU[5]	0.50	0.11	0.51	0.13	0.11	0.82	0.19
Phased LSTM[8]	0.51	0.12	0.48	0.13	0.09	0.77	0.16
IndRNN[9]	0.64	0.15	0.64	0.13	0.11	0.71	0.19
LNN[10]	0.71	0.12	0.71	0.11	0.15	0.71	0.24
			medium 60%				
Vanilla[6]	0.58	0.12	0.59	0.12	0.11	0.83	0.20
LSTM[4]	0.49	0.14	0.49	0.13	0.09	0.70	0.16
GRU[5]	0.57	0.12	0.51	0.13	0.09	0.73	0.17
Phased LSTM[8]	0.67	0.12	0.66	0.13	0.07	0.44	0.13
IndRNN[9]	0.43	0.13	0.41	0.13	0.08	0.74	0.14
LNN[10]	0.64	0.11	0.66	0.11	0.12	0.78	0.20

Table 2: Impact of Dataset Size on Model Performance (ozone dataset)

Sub-RQ2 (How do Liquid Neural Networks and RNN-based Neural Networks perform when trained on datasets of different sizes (small vs. large datasets)?) was addressed by evaluating model performance on subsets of the Ozone dataset, specifically 2%, 20%, and 60% of the whole dataset, to simulate very-small-data, small-data, and medium-data regimes, respectively. This approach follows

the methodology outlined in Section 3.2, with results presented in Table 2 for classification metrics across these dataset sizes.

In the very small (2%) dataset regime, LNN achieves the highest validation and test accuracies (0.60 and 0.65, respectively) and competitive CCE (0.12), outperforming all RNN-based models. Phased LSTM follows with a validation accuracy of 0.40 and the lowest CCE (0.07), indicating strong performance in small-data settings, likely due to its time-gated mechanism suited for sparse data. LSTM and GRU struggle significantly, with low accuracies (0.13 and 0.27, respectively), reflecting challenges in the small-data regime, as noted by Bornschein et al.[11]. In the small (20%) dataset regime, LNN continues to lead with validation and test accuracies of (0.71) and a CCE of (0.11 and 0.12), followed by IndRNN (0.64 accuracy). GRU and LSTM show improved performance but remain below LNN. In the medium (60%) dataset regime, Phased LSTM and LNN perform comparably (0.66 and 0.67 test accuracy), while IndRNN and LSTM show reduced performance (0.41 and 0.49, respectively). These results highlight LNN's robustness across varying dataset sizes, supporting the hypothesis that LNNs adapt effectively to limited data due to their continuous-time dynamics[10].

4.3 Domain-Based Comparison (Sub-RQ3)

Sub-RQ3 (How do different models perform across various time series forecasting domains (e.g., financial markets, weather, energy consumption)?) was addressed by evaluating model performance across the Gesture, Occupancy, Traffic, and Power datasets, representing human activity, environmental, transportation, and energy domains, respectively. The experimental procedure closely follows the evaluation methodology proposed by Hasani et al.[10], which benchmarks neural network performance across diverse time series datasets. The results are presented in Table 3, with accuracy reported for Gesture and Occupancy (classification tasks) and squared error (MSE) for Traffic and Power (regression tasks).

Datasets	metrics	Vanilla	LSTM	GRU	Phased LSTM	IndRNN	LNN
Gesture	(accuracy)	56.60%	59.13%	60.66%	57.05%	52.95%	63.06%
Occupancy	(accuracy)	90.13%	93.39%	92.16%	92.85%	92.10%	93.67%
Traffic	(squared error)	0.171	0.158	0.160	0.189	0.342	0.147
Power	(squared error)	0.018	0.017	0.018	0.022	0.018	0.016

Table 3: Model Performance Across Different Domains

For the Gesture dataset, LNN achieves the highest accuracy (63.06%), followed by GRU (60.66%) and LSTM (59.13%), indicating LNN's effectiveness in handling human activity data with potential non-stationary patterns. In the Occupancy dataset, LNN again leads with 93.67% accuracy, closely followed by LSTM (93.39%) and Phased LSTM (92.85%), suggesting robust performance across models in environmental settings with more stable patterns. For the Traffic dataset, LNN achieves

the lowest squared error (0.147), outperforming LSTM (0.158) and GRU (0.160), with IndRNN performing poorly (0.342), likely due to challenges with transportation data's complex temporal dynamics. Similarly, in the Power dataset, LNN records the lowest squared error (0.016), followed by LSTM (0.017), highlighting LNN's superior performance in energy consumption forecasting.

4.4 Effect of Loss Functions (Sub-RQ4)

Sub-RQ4 (What impact does the choice of loss function have on the performance of Liquid Neural Networks compared to RNN-based Neural Networks?) was addressed by training all models on the Gesture dataset using four loss functions: Mean Squared Error (MSE), Mean Absolute Error (MAE), Huber, and Log-Cosh, as described in Section 3.5. The performance is evaluated using classification accuracy, with results presented in Table 4.

Loss Functions	Vanilla	LSTM	GRU	Phased LSTM	IndRNN	LNN
MSE	57.60%	58.51%	60.03%	54.44%	53.98%	64.41%
MAE	54.03%	60.87%	59.97%	51.25%	50.59%	50.52%
Huber	59.09%	61.67%	61.22%	55.38%	55.10%	68.40%
Log-Cosh	59.20%	58.82%	62.02%	56.03%	54.51%	64.13%

Table 4: Effect of Different Loss Functions on Model Performance (gesture dataset)

LNN achieves the highest accuracy with the Huber loss function (68.40%), followed by MSE (64.41%) and Log-Cosh (64.13%), but performs poorly with MAE (50.52%), suggesting sensitivity to loss function choice. GRU and LSTM perform consistently across loss functions, with GRU peaking at 62.02% (Log-Cosh) and LSTM at 61.67% (Huber). Vanilla RNN, Phased LSTM, and IndRNN show lower accuracies across all loss functions, with Huber and Log-Cosh generally yielding better results than MAE. These findings align with Jadon et al.[15], who noted that loss function choice significantly impacts performance and highlights LNN's superior performance with Huber loss, likely due to its robustness to outliers in non-stationary data. The poor performance of MAE across all models suggests it is less suitable for classification tasks on the Gesture dataset, reinforcing the need for careful loss function selection, as emphasized by Raximov et al.[16].

5 Discussion

5.1 Analysis of Main Findings

For Sub-RQ1, LNNs demonstrated strong performance in classification tasks on the Ozone dataset, particularly in achieving a low Categorical Cross-Entropy, though GRU outperformed them in regression tasks. This indicates LNNs' suitability for non-stationary data classification, as noted by Hasani et al.[10]. For Sub-RQ2, LNNs consistently outperformed all RNN models across small-data regimes (2% and 20% Ozone subsets) and remained competitive in larger datasets (60%), highlighting their robustness and adaptability in data-scarce scenarios [11]. Addressing Sub-RQ3, LNNs demonstrated superior performance across diverse domains, leading to accuracy on the Gesture

and Occupancy datasets and achieving the lowest errors on the Traffic and Power datasets, thereby confirming their cross-domain adaptability [21]. For Sub-RQ4, LNNs achieved the highest accuracy with Huber loss on the Gesture dataset, but their performance varied significantly with loss function choice, underscoring the importance of optimization for their effectiveness [15].

In addressing the main research question, "How do Liquid Neural Networks compare to RNN-based Neural Networks (Vanilla RNN, LSTM, GRU, Phased LSTM, and IndRNN) in time series forecasting tasks?" the conclusions indicate that LNNs generally outperform RNN-based models. Their strengths in classification, small-data robustness, cross-domain adaptability, and performance with optimized loss functions make them a better choice than RNN variants, particularly in non-stationary and data-limited contexts, despite GRU's edge in regression tasks.

Additionally, on the traffic and Power datasets, which contain over 10000 and 16,000 entries, respectively, IndRNN exhibited superior training efficiency, being faster than every other model, highlighting a computational advantage for IndRNN in large-scale datasets.

5.2 Limitations

First, the use of default hyperparameters without tuning, as specified in Section 3.3, may not fully optimize the performance of each model. Hoque and Aljamaan [20] demonstrated that hyperparameter optimization significantly enhances forecasting accuracy, suggesting that tailored configurations could alter the comparative outcomes, particularly for LNNs, which exhibited sensitivity to loss function choice in Section 4.4.

Second, the dataset selection, primarily sourced from Hasani et al.[10], lacks representation of financial time series and other types. This limits the assessment of LNNs in critical application areas such as stock price forecasting.

Third, the implementation of LNNs in the older TensorFlow 1.15 version in Google Colab and RNNs in PyTorch in PyCharm, as detailed in Section 3.1, may introduce framework-specific biases, despite using the original author's codes.

Finally, the study focuses on a limited number of loss functions (MSE, MAE, Huber, and Log-Cosh) and a limited selection of the latest RNN architectures (Phased LSTM and IndRNN).

5.3 Novel Contributions

First, the thesis provides the first detailed benchmark comparing LNNs with a full spectrum of RNN-based architectures across multiple domains (Ozone, Gesture, Occupancy, Power, and Traffic), dataset sizes (2%, 20%, and 60% Ozone subsets), evaluation metrics (regression and classification), and loss functions (MSE, MAE, Huber, Log-Cosh). This evaluation fills a significant research gap noted by Hasani et al.[10] and Rubanova et al.[18], who highlighted the limited comparative studies involving LNNs.

Second, the study introduces a methodology for assessing model performance in small-data regimes, demonstrating LNNs' superior adaptability with leading accuracies across the 2% and 20% Ozone subsets. This extends the findings of Bornschein et al.[11] on model selection challenges in data-scarce contexts. Moreover, the thesis reveals a notable computational insight with IndRNN's superior training efficiency in larger datasets.

Third, the analysis of loss function impacts on LNNs provides new insights into optimizing continuous-time models, including: (1) the Huber loss function significantly enhances LNN performance, achieving the highest classification accuracy; (2) LNNs exhibit sensitivity to the MAE loss function, resulting in poor performance for classification tasks, indicating MAE's unsuitability for such scenarios; and (3) MSE and Log-Cosh loss functions yield competitive LNN accuracies, suggesting their viability for continuous-time models.

Collectively, these contributions establish a foundational benchmark for evaluating LNNs against RNN-based models, offering practical insights for their application in dynamic and data-limited forecasting tasks and paving the way for further advancements in the field.

5.4 Future Research

First, future studies should explore hyperparameter optimization to improve the performance of LNNs and RNN variants, as the current study used default settings to train every model(Section 3.3).

Second, expanding the dataset scope to include financial time series and others types, such as stock price data, would address the limitation of domain coverage (Section 5.2).

Finally, integrating LNNs with emerging architectures, such as Graph Neural Networks (GNNs), could enhance forecasting capabilities by capturing spatial-temporal relationships, as suggested by Kumbure et al.[22] and Jin et al.[23]. This hybridization may improve performance on complex datasets, such as those involving network-structured time series data, which are increasingly relevant in domains like transportation and energy.

These research directions aim to refine the understanding and deployment of LNNs, addressing current limitations and leveraging recent advancements to position them as a leading solution in time series forecasting.

6 Conclusions

This thesis has systematically compared Liquid Neural Networks (LNNs) with RNN-based architectures (Vanilla RNN, LSTM, GRU, Phased LSTM, and IndRNN) in time series forecasting tasks. The results demonstrated that LNNs generally outperform traditional RNN-based models, especially in classification tasks, small-data regimes, and across diverse domains. While GRUs exhibited stronger performance in regression metrics. Moreover, LNNs showed greater robustness in small and medium datasets, where RNN models like Vanilla RNN and LSTM struggled. The loss function analysis further highlighted that Huber loss optimized LNN performance most effectively,

achieving the highest classification accuracy on the Gesture dataset.

In conclusion, Liquid Neural Networks (LNNs) present a strong alternative to conventional RNN models, showing comparable or even superior performance in various situations discussed in the study. Future research should aim to enhance their efficiency through hyperparameter optimization, extend their application to a broader array of domains, and investigate hybrid approaches combining them with advanced architectures such as Graph Neural Networks.

References

[1] Patricia M West, Patrick L Brockett, and Linda L Golden. A comparative analysis of neural networks and statistical methods for predicting consumer choice. *Marketing Science*, 16(4):370–391, 1997.

[2] J. D. Restrepo and D. Gaviria. Comparative analysis of artificial neural networks and statistical models applied to demand forecasting. In *2022 IEEE International Conference on Automation/XXV Congress of the Chilean Association of Automatic Control (ICA-ACCA)*, pages 1–6, 2022.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[4] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[5] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[6] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[7] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pages 1310–1318, 2013.

[8] D. Neil, M. Pfeiffer, and S.-C. Liu. Phased lstm: Accelerating recurrent network training for long or event-based sequences. *Advances in Neural Information Processing Systems 29 (NIPS 2016)*, pages 3882–3890, 2016.

[9] S. Li, W. Li, C. Cook, C. Zhu, and Y. Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. *arXiv preprint arXiv:1803.04831*, 2018.

[10] R. Hasani, M. Lechner, A. Amini, D. Rus, and R. Grosu. Liquid time-constant networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35:7657–7666, 2021.

[11] J. Bornschein, F. Visin, and S. Osindero. Small data, big decisions: Model selection in the small-data regime. *Proceedings of the 37th International Conference on Machine Learning*, 119:1035–1044, 2020.

[12] J. Gütter, A. Kruspe, X. X. Zhu, and J. Niebling. Impact of training set size on the ability of deep neural networks to deal with omission noise. *Frontiers in Remote Sensing*, 3:932431, 2022.

[13] Lars Vidar Magnusson, J. Roland Olsson, and Chau Thi Thuy Tran. Recurrent neural networks for oil well event prediction. *IEEE Intelligent Systems*, 38(2):73–80, 2023.

[14] T. Baihaqi, M. A. Sugiyarto, R. P. Daksa, F. I. Kurniadi, M. Fakhruddin, and H. Erandi. Unveiling the precision of deep learning models for stock price prediction: A comparative analysis of bi-lstm, lstm, and gru. *2023 International Conference on Converging Technology in Electrical and Information Engineering (ICCTEIE)*, pages 61–64, 2023.

[15] Aryan Jadon, Avinash Patil, and Shruti Jadon. A comprehensive survey of regression based loss functions for time series forecasting. *arXiv preprint arXiv:2211.02989*, 2022.

[16] Nodir Raximov, Jura Kuvandikov, and Dilmurod Khasanov. The importance of loss function in artificial intelligence. *2022 International Conference on Information Science and Communications Technologies (ICISCT)*, pages 1–3, 2022.

[17] Srijani Maity and Snehendu Saha. A theoretical perspective and experimental evaluation of the extensive analysis of loss functions in machine learning and deep learning. In *2024 IEEE 1st Karachi Section Humanitarian Technology Conference (KHTC)*, pages 1–6, 2024.

[18] Yulia Rubanova, Alexander Remorov, Akosua Busia, Alex Shon, Mike Hughes, Lav R. Varshney, and Rosalind Picard. A novel uncertainty-aware liquid neural network for noise-resilient time series forecasting and classification. *Neural Networks*, 177:106023, 2024.

[19] M. Zhou, S. Ghareeb, Z. U. Shamszaman, and J. Mustafina. Enhancing stock price forecasting: Integrating supply chain factors into lstm models and comparative performance analysis. *2023 16th International Conference on Developments in eSystems Engineering (DeSE)*, pages 253–257, 2023.

[20] Kazi Emon Hoque and Hamed Aljamaan. Impact of hyperparameter tuning on machine learning models in stock price forecasting. *IEEE Access*, 9:163815–163830, 2021.

[21] Ciaran McCaffrey, Alan McKee, and Philip Morrow. A comparative analysis of state-of-the-art time series forecasting algorithms. *Proceedings of the 2022 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2:89–96, 2022.

[22] M. M. Kumbure, A. W. Malik, D. M. de Matos, H. A. B. F. de Oliveira, T. O. Ayodele, M. Ghasemi, A. P. Engelbrecht, and A. A. Heikalabad. A systematic review on graph neural network-based methods for stock market forecasting. *ACM Computing Surveys*, 56(10):252, 2024.

[23] M. Jin, J. Wang, X. Liu, Y. Zhang, L. Wu, X. Lin, and L. He. A survey on graph neural networks for time series: Forecasting, classification, imputation, and anomaly detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):9440–9458, 2024.