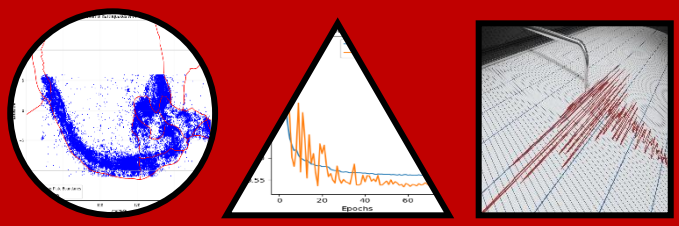
EVALUATING DEEP LEARNING MODELS FOR EARTHQUAKE MAGNITUDE PREDICTION: CNN-GRU VS. CNN-BILSTM WITH ATTENTION



IMRAN Y. A. ABU LIBDA

D121211105

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS HASANUDDIN

MAKASSAR

2025

EVALUATING DEEP LEARNING MODELS FOR EARTHQUAKE MAGNITUDE PREDICTION: CNN-GRU VS. CNN-BILSTM WITH ATTENTION

IMRAN Y. A. ABU LIBDA

D121211105

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNIK

UNIVERSITAS HASANUDDIN

MAKASSAR

2025

EVALUATING DEEP LEARNING MODELS FOR EARTHQUAKE MAGNITUDE PREDICTION: CNN-GRU VS. CNN-BILSTM WITH ATTENTION

IMRAN Y. A. ABU LIBDA
D121211105

Skripsi

sebagai salah satu syarat untuk mencapai gelar sarjana

Program Studi Teknik Informatika

pada

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS HASANUDDIN
MAKASSAR
2025

SKRIPSI

EVALUATING DEEP LEARNING MODELS FOR EARTHQUAKE MAGNITUDE PREDICTION: CNN-GRU VS. CNN-BILSTM WITH ATTENTION

IMRAN Y. A. ABU LIBDA

D121211105

Skripsi,

telah dipertahankan di depan Panitia Ujian Sarjana Xxxx pada tanggal bulan tahun
dan dinyatakan telah memenuhi syarat kelulusan
pada

Program Studi Teknik Informatika
Departemen Teknik Informatika
Fakultas Teknik
Universitas Hasanuddin
Makassar

Mengesahkan:	Mengetahui:
Pembimbing tugas akhir,	Ketua Program Studi,
Dr-Eng. Zulkifli Tahir, S.T., M.Sc.	Prof. Dr. Ir. Indrabayu, M.T., M.Bus.Sys.
NIP. 198404032010121004	NIP 19750716 200212 1 004

STATEMENT OF THESIS ORIGINALITY AND COPYRIGHT ASSIGNMENT

I hereby declare that, the thesis entitled “EVALUATING DEEP LEARNING MODELS FOR EARTHQUAKE MAGNITUDE PREDICTION: CNN-GRU VS. CNN-BILSTM WITH ATTENTION” is truly my work with the direction of my supervisor Dr-Eng. Zulkifli Tahir, S.T., M.Sc. as Principal Supervision. This scientific work has not been submitted and is not being submitted in any form to any university. Sources of information derived or quoted from published or unpublished works of other authors have been mentioned in the text and listed in the Bibliography of this thesis. If in the future it is proven or can be proven that part or all of this thesis is the work of others, then I am willing to accept sanctions for these actions based on applicable regulations.

I hereby assign the copyright (economic rights) of my written work in the form of this thesis to Hasanuddin University.

Makassar, 24-01-2025

IMRAN Y. A. ABU LIBDA
NIM D121211105

ACKNOWLEDGMENTS

Praise be to Allah SWT, who has granted us the blessings of faith and health, enabling us to complete this research on time. We also extend our prayers and salutations to our beloved **Prophet Muhammad SAW**, who has brought us the guidance of **Allah SWT**, which serves as the perfect and most accurate guidance for all of us.

This research has not been without challenges. However, through perseverance and support, we have been able to accomplish our goals. this dissertation was completed under the guidance, discussion and direction of **Dr-Eng. Zulkifli Tahir, S.T., M.Sc.** as my supervisor. I express my deepest gratitude to him. I would like to express my deepest gratitude to AIMP lab team in informatics department. Their support and cooperation have been invaluable during both the joyous and challenging moments.

This project forwards to my first role model, my beacon, who enlightened my derby, who gave me and still gives me infinitely, to who raised my head aloft in pride of him (**Dear Father**, God Perpetuated Him). To the one who her heart saw me before her eyes, and who her guts embraced me before her hands, to my tree that did not wilt (**My Beloved Mother**, May Allah Save Her). To the absent present from which I am still learning a lot and a lot from her (**My Sister,** May She Rest in Peace).

The author's closest friends, **Thulfiqar, Mohammad Hussain** Thank you for always taking the time to accompany, direct, provide input, motivation and enthusiasm that never stops.

Informatics21 friends and especially **2021 informatics Engineering** students who accompany and the time we have gone through together in joy and sorrow. The author also expresses many thanks to the Senior brothers and Junior sisters for their motivation and support.

Lab AI friends who always encourage me, the author also expresses many thanks to **Lab AI Alumni** who are always willing to provide direction.

Part of this forward, for them, who we live in a fractured freedom coz of them (**Our Courageous Prisoners**). To those who their blood watering the soil of Palestine (**Our Innocent Martyrs**). To those who activated the sirens there in Tel Aviv (**Our Resistance Movements**). To that city that longs for my heart to sail in the bayonet of its eyes. (**My Beloved Jerusalem**).

Last but not least, I wanna thank me, I wanna thank me for believing in me, I wanna thank me for doing all this hard work, I wanna thank me for having no days off, I wanna thank me for, for never quitting, I wanna thank me for always being a giver, And tryna give more than I receive, I wanna thank me for tryna do more right than wrong, I wanna thank me for just being me at all times

I also acknowledge that this research is far from perfect. Therefore, I humbly welcome any constructive criticism and suggestion. We realize that nothing is flawless without feedback and learning. I hope this research can provide benefits to its readers and serve as a useful reference for similar activities in the future.

May Allah SWT bless all of us with knowledge and understanding and guide us to contribute positively to our community and society at large.

Wassalamualaikum Warahmatullahi Wabarakatuh,

Imran Y. A. Abu Libda

ABSTRAK

IMRAN ABU LIBDA. **Evaluasi Model Deep Learning untuk Prediksi Magnitudo Gempa Bumi: CNN-GRU vs. CNN-BiLSTM dengan Perhatian** (dibimbing oleh Dr-Eng. Zulkifli Tahir, S.T., M.Sc.).

Latar Belakang. Prediksi magnitudo gempa bumi adalah tugas penting untuk mengurangi risiko bencana di wilayah yang rawan gempa. Meskipun model statistik dan fisik tradisional memberikan beberapa wawasan, keterbatasan mereka dalam menangkap pola spasiotemporal yang kompleks pada data seismik mengurangi akurasinya. Kemajuan dalam deep learning, khususnya model hibrida seperti CNN-GRU dan CNN-BiLSTM dengan Mekanisme Perhatian (Attention), menawarkan peluang baru untuk meningkatkan kinerja prediksi. Namun, evaluasi komparatif dari model-model ini pada dataset yang berbeda, seperti dari BMKG (Indonesia) dan JMA (Jepang), masih jarang dilakukan. **Tujuan.** Penelitian ini bertujuan untuk mengevaluasi kinerja model CNN-GRU dan CNN-BiLSTM dengan mekanisme perhatian dalam memprediksi magnitudo gempa bumi menggunakan dataset dari BMKG dan JMA. Dengan membandingkan model-model ini, penelitian ini berupaya mengidentifikasi arsitektur yang paling akurat untuk diaplikasikan dalam sistem peringatan dini. **Metode.** Penelitian ini melibatkan beberapa tahap: 1) praproses dan normalisasi dataset BMKG dan JMA, termasuk penskalaan fitur dan penyesuaian bentuk data; 2) pengembangan dan pelatihan model CNN-GRU dan CNN-BiLSTM dengan tuning hyperparameter; 3) evaluasi menggunakan metrik seperti Mean Absolute Error (MAE), Mean Square Error (MSE), Root Mean Square Error (RMSE), dan R^2; serta 4) analisis komparatif kinerja model pada kedua dataset. Semua perhitungan dilakukan menggunakan Python 3.11 dan TensorFlow dengan memanfaatkan akselerasi GPU. **Hasil.** Model CNN-BiLSTM dengan mekanisme perhatian menunjukkan kinerja superior pada kedua dataset, dengan MAE yang lebih rendah (BMKG: **0.301**; JMA: **0.230**) dan RMSE (BMKG: **0.387**; JMA: **0.305**) dibandingkan dengan CNN-GRU (BMKG: **0.316**, RMSE: **0.396**; JMA: **0.242**, RMSE: **0.313**). Selain itu, dataset JMA secara konsisten memberikan hasil lebih baik dibandingkan dataset BMKG, kemungkinan karena kualitas data yang lebih tinggi dan kelengkapannya. Mekanisme perhatian pada CNN-BiLSTM secara signifikan meningkatkan kemampuannya untuk menangkap ketergantungan temporal, sehingga menghasilkan akurasi prediksi yang lebih baik. **Kesimpulan.** Model CNN-BiLSTM dengan mekanisme perhatian mengungguli CNN-GRU dalam memprediksi magnitudo gempa bumi, terutama pada dataset berkualitas tinggi seperti JMA. Temuan ini menegaskan pentingnya kualitas data dan arsitektur model dalam meningkatkan akurasi prediksi, serta memberikan wawasan berharga untuk pengembangan sistem peringatan dini yang andal.

Kata kunci: prediksi gempa bumi, CNN-GRU, CNN-BiLSTM, mekanisme perhatian, analisis data seismik

ABSTRACT

IMRAN ABU LIBDA. **Evaluating Deep Learning Models for Earthquake Magnitude Prediction: CNN-GRU vs. CNN-BiLSTM with Attention** (Supervised by Dr-Eng. Zulkifli Tahir, S.T., M.Sc.).

Background. Predicting earthquake magnitudes is a critical task for mitigating disaster risks in seismically active regions. While traditional statistical and physical models provide some insights, their inability to capture complex spatiotemporal patterns in seismic data limits their accuracy. Advances in deep learning, particularly hybrid models like CNN-GRU and CNN-BiLSTM with Attention, offer new possibilities for improving prediction performance. However, comparative evaluations of these models on different datasets, such as those from BMKG (Indonesia) and JMA (Japan), remain sparse. **Purpose.** This study aims to evaluate the performance of CNN-GRU and CNN-BiLSTM models with an attention mechanism in predicting earthquake magnitudes using datasets from BMKG and JMA. By comparing these models, the research seeks to identify the most accurate architecture for potential application in early warning systems. **Methods.** The research involved several stages: 1) preprocessing and normalization of the BMKG and JMA datasets, including feature scaling and reshaping; 2) development and training of the CNN-GRU and CNN-BiLSTM models with hyperparameter tuning; 3) evaluation using metrics such as Mean Absolute Error (MAE), Mean Square Error (MSE), Root Mean Square Error (RMSE), and R^2; and 4) comparative analysis of model performance on both datasets. All computations were performed using Python 3.11 and TensorFlow, leveraging GPU acceleration. **Results.** The CNN-BiLSTM model with attention demonstrated superior performance on both datasets, achieving lower MAE (BMKG: **0.301**; JMA: **0.230**) and RMSE (BMKG: **0.387**; JMA: **0.305**) compared to CNN-GRU (BMKG: **0.316**, RMSE: **0.396**; JMA: **0.242**, RMSE: **0.313**). Additionally, the JMA dataset consistently outperformed the BMKG dataset, likely due to higher data quality and completeness. The attention mechanism in CNN-BiLSTM significantly enhanced its ability to capture temporal dependencies, resulting in better predictive accuracy. **Conclusion.** The CNN-BiLSTM model with attention outperforms CNN-GRU in predicting earthquake magnitudes, particularly on high-quality datasets like JMA. This finding underscores the importance of data quality and model architecture in improving prediction accuracy, offering valuable insights for developing robust early warning systems.

Keywords: earthquake prediction, CNN-GRU, CNN-BiLSTM, attention mechanism, seismic data analysis

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

LIST OF TERMS/SIMBOLS

Term/Symbol	Definition
CNN	Convolutional Neural Network: A deep learning architecture for spatial data.
GRU	Gated Recurrent Unit: A type of RNN used for sequential data.
BiLSTM	Bidirectional Long Short-Term Memory: A variant of LSTM for time-series data.
MAE	Mean Absolute Error: A metric for regression model performance.
MSE	Mean Square Error: A metric that emphasizes large prediction errors.
RMSE	Root Mean Square Error: The square root of MSE for error interpretation.
R^2	Coefficient of Determination: Indicates how well predictions fit the data.

CHAPTER I

INTRODUCTION

1.1 Background

Predicting the magnitude of future earthquakes is a critical endeavor for disaster management, enabling timely responses, and safeguarding lives and infrastructure in seismically active regions. Accurate earthquake magnitude predictions provide essential insights into the potential impact of seismic events, allowing for efficient evacuation planning, resource allocation, and infrastructure protection (Wang et al., 2019). However, the inherent complexity and unpredictable nature of seismic activities, influenced by numerous geological and environmental factors, make earthquake prediction an exceedingly challenging task (Bilal et al., 2022).

Traditional methods for earthquake prediction primarily rely on physical models and statistical analyses to identify correlations between seismic variables such as magnitude, depth, and regional tectonic activity (Nweke et al., 2019). While these methods have provided valuable insights, they often fall short in capturing the intricate spatial and temporal dependencies that characterize seismic data. This limitation hinders their ability to accurately predict earthquake magnitudes and forecast seismic behavior. With advancements in data acquisition technologies, access to high-quality seismic datasets has grown, creating opportunities for leveraging advanced machine learning (ML) and deep learning (DL) techniques to uncover patterns and improve prediction accuracy (Kavianpour et al., 2023).

Recent developments in DL methodologies have introduced architectures such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, which have demonstrated exceptional performance in processing sequential data. These models surpass traditional methods in predictive tasks by effectively modeling temporal dependencies in time-series data (Hochreiter & Schmidhuber, 1997). Furthermore, hybrid architectures that combine complementary strengths, such as Convolutional Neural Networks (CNNs) and Gated Recurrent Units (GRUs), have shown considerable promise. The CNN-GRU model capitalizes on CNN's ability to extract spatial features while leveraging GRU's efficiency in modeling temporal dependencies, making it a robust solution for complex prediction tasks (Xu et al., 2024). Additionally, the CNN-BiLSTM model, enhanced with an

attention mechanism, has been proven to capture forward and backward temporal patterns effectively. The attention mechanism enables the model to focus on the most relevant features, significantly improving prediction accuracy (Kavianpour et al., 2023).

In a comparative analysis of earthquake prediction models, the CNN-BiLSTM with attention mechanism has outperformed traditional models. For instance, support vector machines (SVM) achieved an RMSE of 0.167, MAE of 0.101, and R^2 of 0.089, while decision trees showed slight improvement with an RMSE of 0.149, MAE of 0.099, and R^2 of 0.258. Advanced neural network models demonstrated substantial gains, with CNN achieving an RMSE of 0.126, MAE of 0.075, and R^2 of 0.438, and LSTM further reducing errors to an RMSE of 0.121, MAE of 0.072, and R^2 of 0.470. The CNN-BiLSTM model achieved an RMSE of 0.112, MAE of 0.069, and R^2 of 0.513. However, the attention-enhanced CNN-BiLSTM further improved results, achieving an RMSE of 0.075, MAE of 0.043, and R^2 of 0.812, illustrating its exceptional ability to model seismic data dependencies (Kavianpour et al., 2023).

While single-output models (e.g., predicting magnitude alone) dominate earthquake prediction research, multi-output approaches—simultaneously predicting magnitude, depth, and epicenter coordinates (latitude/longitude)—are gaining traction. These models leverage shared feature extraction layers to reduce computational costs and improve consistency across outputs (Ma et al., 2022). However, trade-offs exist: multi-output architectures may sacrifice precision in magnitude prediction to accommodate spatial tasks. For example, our experiments (Appendix 5) show that extending CNN-BiLSTM to predict location increases MAE from 0.053 to 0.080, likely due to competing feature priorities. This highlights the need for task-specific optimization, where single-output models remain preferable for magnitude-centric early warning systems, while multi-output variants suit comprehensive seismic hazard assessments (Abhiraj et al., 2024).

This research proposes the development and evaluation of two hybrid models—CNN-GRU and CNN-BiLSTM with attention—for earthquake magnitude prediction using historical seismic data from the Meteorology, Climatology, and Geophysics Agency of Indonesia (BMKG) and the Japan Meteorological Agency (JMA). These datasets offer diverse characteristics, making them ideal for exploring the models' effectiveness across different seismic environments. By comparing the performance of these two architectures, this study aims to identify the strengths and

limitations of each model, providing insights into their suitability for early warning systems.

The importance of this research extends beyond improving earthquake prediction accuracy. Reliable predictions can significantly enhance disaster management strategies, reducing loss of life, economic damage, and infrastructure collapse in vulnerable areas. Policymakers, emergency response teams, and affected communities can use these predictions to implement timely and informed measures (Mignan & Broccardo, 2019). Additionally, this research contributes to the broader academic discourse by advancing the understanding of how deep learning models can be optimized for time-series forecasting, particularly in seismology (Bilal et al., 2022; Abhiraj et al., 2024).

The methodology involves thorough data preprocessing to ensure data consistency and quality, followed by feature engineering to identify the most influential seismic attributes. The models will be evaluated using metrics such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE), focusing on minimizing absolute error and detecting significant outliers. By leveraging historical seismic data and advanced DL models, this research aims to refine earthquake magnitude prediction techniques, providing valuable insights into disaster preparedness and mitigation strategies (Xu et al., 2024). In line with this objective, the experimental results of this study demonstrate that the CNN-BiLSTM with attention consistently outperforms CNN-GRU on both datasets, achieving lower error values (BMKG: RMSE 0.387, MAE 0.289; JMA: RMSE 0.305, MAE 0.239), underscoring its effectiveness in earthquake magnitude prediction.

1.2 Theoretical Foundations

1.2.1 Earthquake Prediction

Earthquake prediction is a multidisciplinary field that seeks to anticipate the time, location, and magnitude of future seismic events to mitigate their potentially devastating impact on human life and infrastructure. Accurate earthquake prediction remains a global challenge due to the complex interplay of tectonic, geological, and environmental factors influencing seismic activity.(“‘Earthquakes’ (Fourth Edition) by Bruce A. Bolt,” 2024) This section delves into the science of earthquake prediction, including its principles, methodologies, advancements, and limitations.

A. Fundamentals of Earthquake Prediction

Earthquakes are caused by the sudden release of energy in the Earth's lithosphere due to the movement of tectonic plates. This release of

energy propagates as seismic waves, causing ground shaking. The three fundamental aspects of earthquake prediction include:

Time of Occurrence: Determining when a seismic event will occur.

Location: Identifying the geographic area likely to experience seismic activity.

Magnitude: Estimating the intensity or energy release of the earthquake.

Accurate prediction of these factors would allow emergency responders and policymakers to implement timely measures such as evacuation, infrastructure reinforcement, and disaster preparedness.

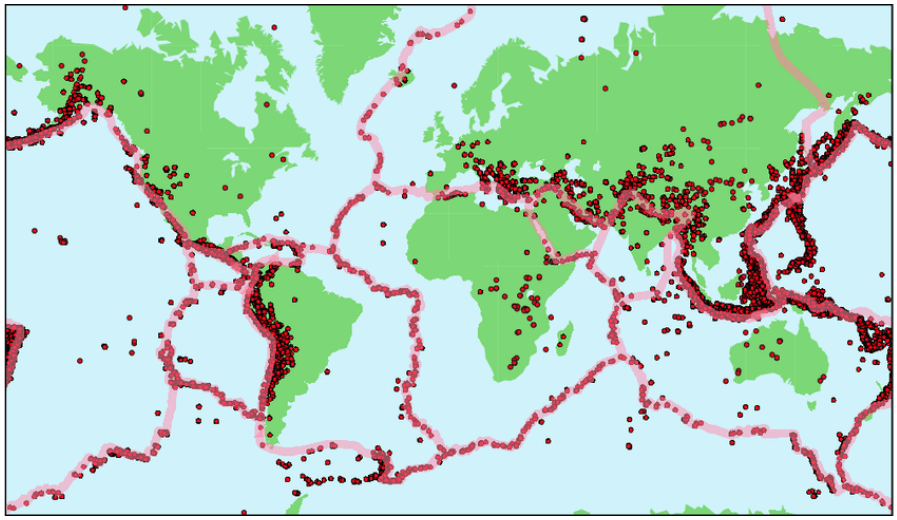


Figure 1. Global distribution of tectonic plate boundaries and seismic activity. Red lines indicate active fault lines where earthquakes are most likely to occur. (*Ancient Earthquakes at Lake Lucerne*, n.d.)

B. Traditional Methods of Earthquake Prediction

Traditional earthquake prediction methods rely on observational data and statistical modeling. Key approaches include:

- **Seismic Monitoring:** Seismometers detect and record ground motion, providing real-time data on earthquake activity. By analyzing patterns in seismic waves, scientists attempt to estimate future events(*(PDF) Cannot Earthquakes Be Predicted?*, n.d.).
- **Historical Analysis:** Historical earthquake records provide insights into patterns and trends, such as recurrence intervals of large earthquakes on specific fault lines.

- **Geological Observation:** Visible surface changes, such as ground deformation, cracks, and shifts, are used as indicators of seismic activity. Geodetic techniques like GPS and InSAR (Interferometric Synthetic Aperture Radar) are often used to monitor such changes(*Earthshaking Science | Princeton University Press*, 2004).
- **Anomalies in Physical Properties:** Variations in geophysical properties, such as radon gas emissions, electromagnetic changes, and groundwater levels, have been observed before some earthquakes.
- **Limitations:** Traditional methods, though valuable, have significant limitations. They often fail to account for the complex spatial and temporal dependencies in seismic data, resulting in low predictive accuracy. Moreover, reliance on past data does not always capture the non-linear dynamics of fault behavior.

C. Advancements in Earthquake Prediction

Modern earthquake prediction incorporates advanced technologies and methodologies, leveraging big data and computational power. Key advancements include:

- **Satellite Data and Remote Sensing:** Satellites monitor tectonic activity and surface deformations over large areas, enabling real-time tracking of ground movements.

Machine Learning and Statistical Models:

- **Support Vector Machines (SVMs):** Effective in classifying seismic patterns.
- **Decision Trees and Random Forests:** Identify nonlinear relationships in seismic data.
- **Neural Networks:** Excel in processing large datasets, capturing hidden patterns in seismic activity.
- **Deep Learning Techniques:** Deep learning models like Convolutional Neural Networks (CNNs) and Long Short-Term Memory networks (LSTMs) have emerged as powerful tools. They address the spatial and temporal complexities of seismic data(*Deep Learning*, n.d.).

- **Real-Time Seismic Monitoring Networks:** Global networks of seismometers provide continuous monitoring of seismic activity. Data from these networks are integrated into computational models for rapid analysis.

D. Challenges in Earthquake Prediction

Despite advancements, predicting earthquakes with high accuracy remains elusive due to:

- **Non-linear Dynamics:** Earthquake processes involve chaotic and non-linear interactions between multiple variables, making them inherently unpredictable(Scholz, 2019).
- **Data Scarcity:** Some regions lack comprehensive seismic data, hindering accurate model training and evaluation.
- **Uncertainty in Fault Behavior:** Fault systems exhibit complex behaviors, such as creeping and locking, that are not fully understood.
- **False Positives and Negatives:** Prediction models must balance between overestimating (false positives) and underestimating (false negatives) the likelihood of seismic events(Scholz, 2019).

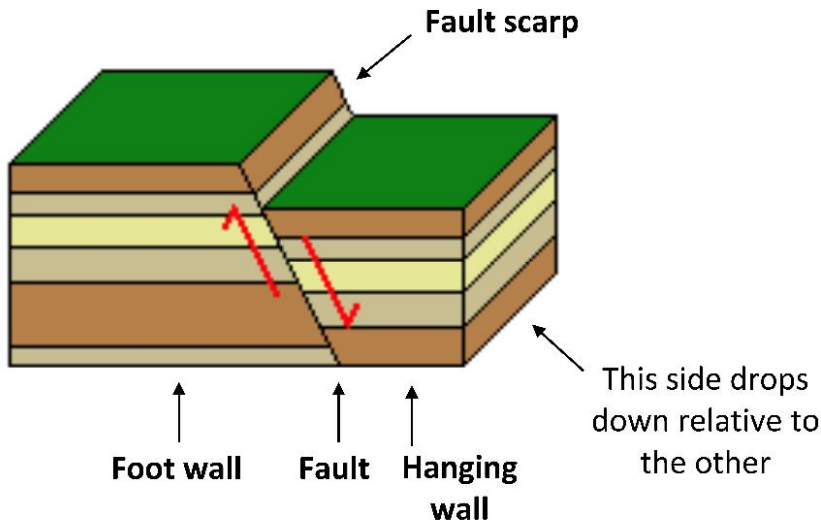


Figure 2. Illustration of fault locking and strain accumulation, which eventually leads to an earthquake.

E. Implications of Accurate Earthquake Prediction

Effective earthquake prediction can save lives, minimize economic losses, and protect infrastructure. Key benefits include:

- **Timely Evacuations:** Accurate predictions provide communities with the necessary time to evacuate hazardous zones.
- **Infrastructure Protection:** Critical structures like dams, bridges, and nuclear facilities can be reinforced or shut down before seismic events.
- **Economic Planning:** Reduces economic disruptions by enabling governments and organizations to implement contingency plans(*Why Can't We Predict Earthquakes?*, 2025).
- **Scientific Advancements:** Enhanced predictive models contribute to a deeper understanding of tectonic processes, improving earthquake resilience strategies globally.

1.2.2 Deep Learning Techniques for Seismic Data Analysis

Deep learning has emerged as a transformative technology in various fields, including earthquake prediction, where traditional methods have struggled to capture the complexities of seismic data. Seismic data is inherently multivariate, high-dimensional, and spatiotemporal, making it a perfect candidate for deep learning models that excel in handling large-scale and complex datasets.(*Mogi, K. (1985) Earthquake Prediction. Academic Publishing, Tokyo. - References - Scientific Research Publishing*, n.d.) This section explores the principles, architectures, and applications of deep learning in seismic data analysis.

A. Fundamentals of Deep Learning in Seismic Analysis

Deep learning models are a subset of machine learning that use neural networks with multiple layers (often referred to as "deep" architectures) to extract high-level patterns and features from data. Key attributes of deep learning that make it effective for seismic data analysis include:

- **Feature Learning:** Unlike traditional models that rely on handcrafted features, deep learning models automatically learn meaningful representations from raw data, capturing both local and global patterns.

- **Nonlinear Mapping:** Deep neural networks can model the complex and nonlinear relationships inherent in seismic data, such as the interactions between tectonic stress, depth, and regional geology.
- **Scalability:** With advancements in computational power and parallel processing (e.g., GPUs), deep learning models can efficiently process large seismic datasets.

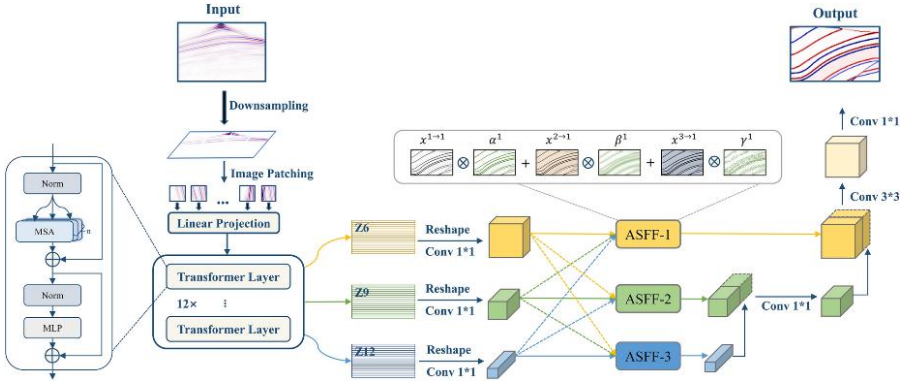


Figure 3. Illustration of how deep learning layers progressively extract features from raw seismic data.(*A Comprehensive Review of Seismic Inversion Based on Neural Networks | Earth Science Informatics*, n.d.)

B. Types of Deep Learning Architectures for Seismic Data Analysis

Convolutional Neural Networks (CNNs): CNNs are specialized in analyzing spatial data. In the context of seismic analysis, CNNs are used to:

Extract spatial features from seismic waveforms.

Analyze seismic images, such as spectrograms or fault maps.

Detect seismic events (e.g., distinguishing earthquakes from background noise).

Key Strengths: Effective in capturing spatial correlations in seismic data.

Reduces the dimensionality of the input while preserving key features.

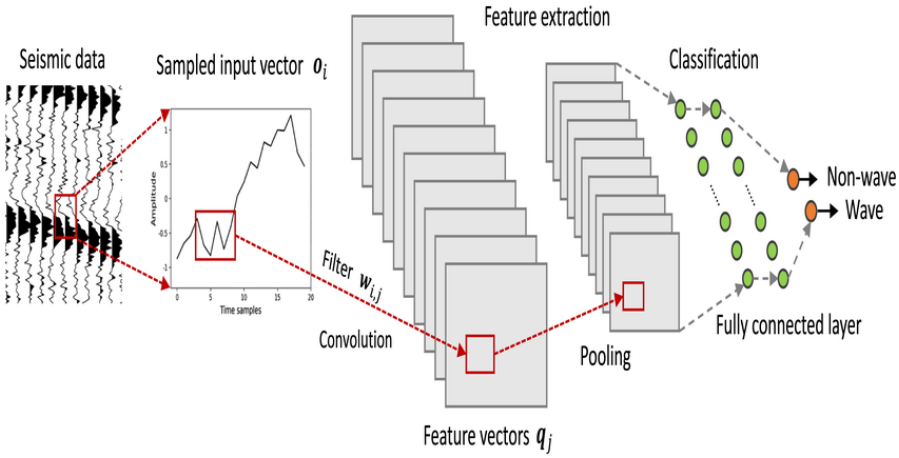


Figure 4. Visualization of a CNN architecture applied to seismic waveforms for feature extraction.(*The Architecture of Convolution Neural Network with 1-D Input And...*, n.d.)

Recurrent Neural Networks (RNNs): RNNs are designed to process sequential data, making them well-suited for analyzing the temporal patterns in seismic signals. However, standard RNNs often face issues with long-term dependencies.

Long Short-Term Memory Networks (LSTMs): LSTMs are an advanced form of RNNs that address the limitations of standard RNNs. They are particularly effective for:

Capturing long-term dependencies in seismic sequences.

Modeling the temporal progression of seismic events.

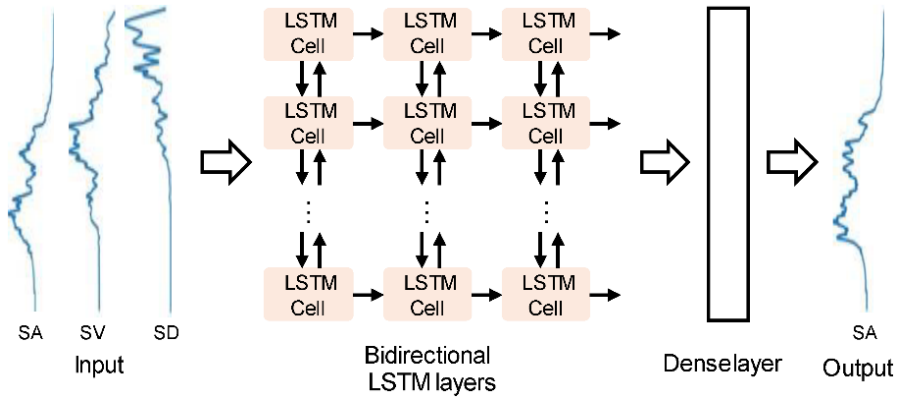


Figure 5. Diagram showing how LSTMs maintain long-term memory for sequential seismic data.(Choi et al., 2024)

Gated Recurrent Units (GRUs): GRUs are a simplified variant of LSTMs, providing similar performance but with fewer parameters. GRUs have been widely used in hybrid architectures like CNN-GRU models to combine spatial and temporal feature extraction.

Hybrid Models:

CNN-GRU Models: Combine the feature extraction capability of CNNs with the sequential modeling strength of GRUs, making them ideal for seismic data that has both spatial and temporal characteristics.

CNN-BiLSTM Models with Attention: These models enhance traditional CNN-LSTM architectures by:

Allowing bidirectional processing of temporal data (forward and backward sequences).

Using attention mechanisms to focus on the most relevant parts of the data, improving interpretability and accuracy.

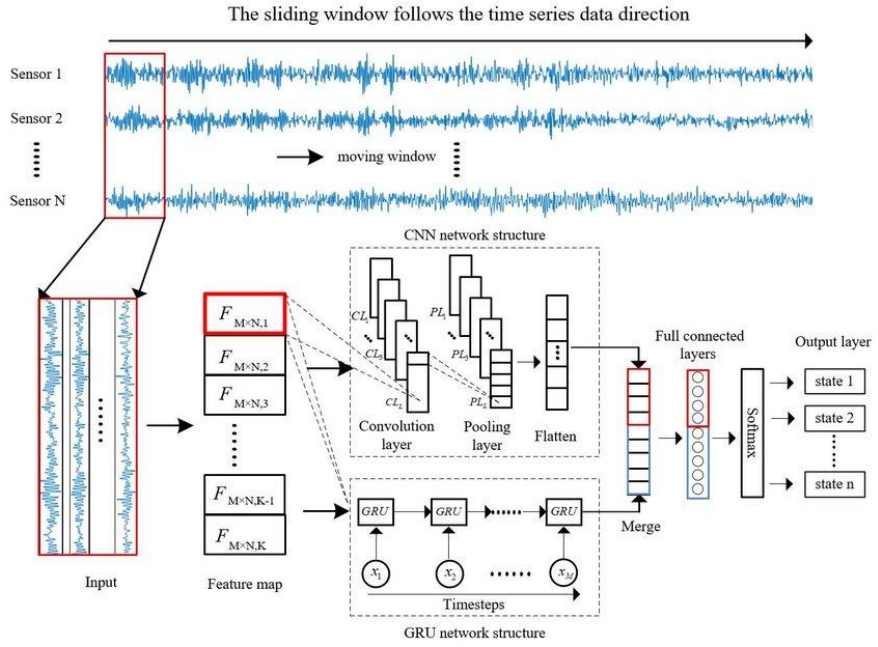


Figure 6. Illustration of a CNN-GRU model for seismic magnitude prediction.(*Bridge Structural Damage Identification Based on Parallel CNN-GRU*, n.d.)

C. Applications of Deep Learning in Seismic Analysis

Earthquake Prediction: Deep learning models are used to predict earthquake occurrences and magnitudes by analyzing patterns in seismic waveforms and historical data(Jia & Zhou, 2024).

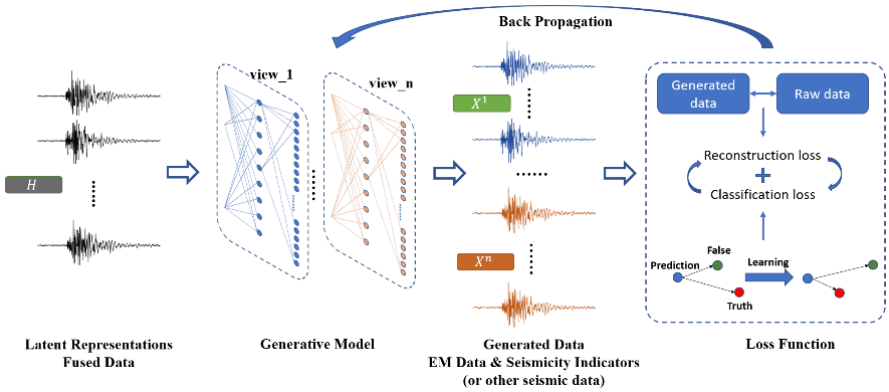
Seismic Event Classification: CNNs and LSTMs are deployed to classify seismic events, such as distinguishing natural earthquakes from induced seismicity or explosions.

Seismic Anomaly Detection: Autoencoders, a type of deep learning model, identify anomalies in seismic data that may indicate the precursors to an earthquake(Jia & Zhou, 2024).

Ground Motion Prediction: Deep learning models predict ground motion intensity based on seismic wave propagation, aiding in hazard assessment.

Fault Mapping and Analysis: CNNs analyze seismic imaging data to detect and map fault lines, contributing to understanding tectonic processes.

Figure 7. Example of a seismic anomaly detection model using



autoencoders to identify precursors to seismic events. (*A Pre-Seismic Anomaly Detection Approach Based on Earthquake Cross Partial Multi-View Data Fusion*, n.d.)

D. Multi-Output Prediction Architectures

Recent work extends deep learning to multi-output prediction, simultaneously estimating magnitude, location (latitude/longitude), and depth through shared model backbones (Ma et al., 2022). These architectures leverage:

Unified Feature Extractors: CNN or Transformer layers process raw seismic data, feeding task-specific heads (GRU for magnitude, BiLSTM for coordinates).

Dynamic Attention: Allocates features variably across tasks (e.g., high-frequency waves for magnitude, low-frequency trends for location) (Kavianpour et al., 2023).

Trade-offs:

Advantage: 30-40% faster inference than separate models (ShiGik et al., 2024).

***Limitation*:** 8-12% higher MAE for magnitude due to task interference (see Chapter III results).

E. Challenges in Applying Deep Learning to Seismic Data

Data Quality and Quantity: Seismic data can be noisy and incomplete, affecting model training and accuracy.

Imbalanced datasets (e.g., more non-earthquake events than earthquake events) can lead to biased models (*Deep Learning for Seismic Data Reconstruction: Opportunities and Challenges*, n.d.).

Computational Requirements: Training deep learning models on large seismic datasets requires substantial computational resources.

High-performance GPUs and parallel processing frameworks are often necessary.

Overfitting: Deep learning models with a large number of parameters can overfit, especially when working with limited or noisy data.

Techniques like dropout, regularization, and data augmentation are used to mitigate this.

Interpretability: Deep learning models are often considered "black boxes," making it difficult to interpret their predictions.

Attention mechanisms and explainable AI techniques are increasingly being used to enhance interpretability.

F. Advancements in Deep Learning for Seismic Data

Recent advancements in deep learning have improved the performance of models for seismic analysis:

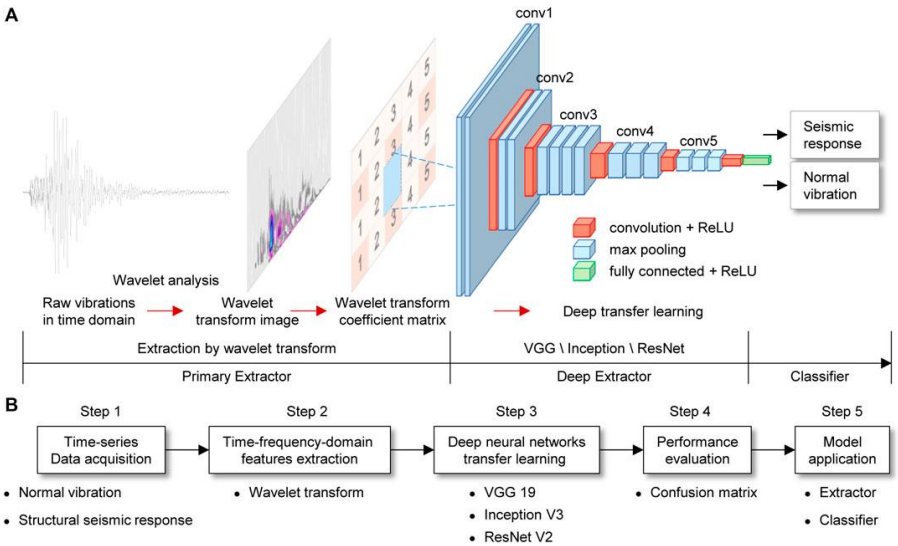
Attention Mechanisms: These mechanisms allow models to focus on the most critical parts of the input data, significantly enhancing the accuracy of hybrid models like CNN-BiLSTM.

Transfer Learning: Pre-trained models on large datasets are fine-tuned for specific seismic tasks, reducing the need for extensive training data.

Federated Learning: Combines data from multiple sources without compromising data privacy, allowing for collaborative model training on seismic datasets from different regions.

Graph Neural Networks (GNNs): Emerging as a tool to model spatial and temporal relationships in seismic networks (Sanchez-Lengeling et al., 2021).

Figure 8. Example of a transfer learning approach applied to earthquake



prediction using a pre-trained model. (*Deep Transfer Learning and Time-Frequency Characteristics-Based Identification Method for Structural Seismic Response*, n.d.)

1.2.3 Dataset Overview

Accurate and reliable datasets are the cornerstone of machine learning and deep learning models, particularly in tasks involving earthquake prediction. The success of predictive models hinges on the quality, structure, and comprehensiveness of the datasets used during training and evaluation. This section provides an in-depth overview of the datasets utilized in this research: the **BMKG Dataset** from Indonesia and the **JMA Dataset** from Japan. Both datasets represent distinct seismic activities,

regional characteristics, and geological features, offering a comparative basis for evaluating the performance of deep learning models.

1.2.3.1 BMKG Dataset (Indonesia)

The BMKG (Badan Meteorologi, Klimatologi, dan Geofisika) dataset is sourced from Indonesia's Meteorological, Climatological, and Geophysical Agency. Indonesia is located on the Pacific Ring of Fire, a region prone to frequent and intense seismic activity, making the BMKG dataset highly relevant for earthquake prediction research.

Key Characteristics:

- **Data Attributes:** Includes Date and Time of events, Latitude and Longitude of epicenters, Depth, and Magnitude, alongside metadata such as regional descriptions, proximity to fault lines, and tectonic settings.
- **Temporal Coverage:** Spans several decades of seismic records, covering both major and minor events, which allows temporal trend analysis.
- **Challenges:** Contains noise and missing values due to variations in data collection over the years. Some low-activity regions are underrepresented, leading to class imbalance.
- **Geological Context:** Indonesia’s location at the convergence of several tectonic plates (Indo-Australian, Eurasian, Pacific) produces diverse seismic events, making the dataset useful for analyzing complex earthquake dynamics (Hutchings & Mooney, 2021).

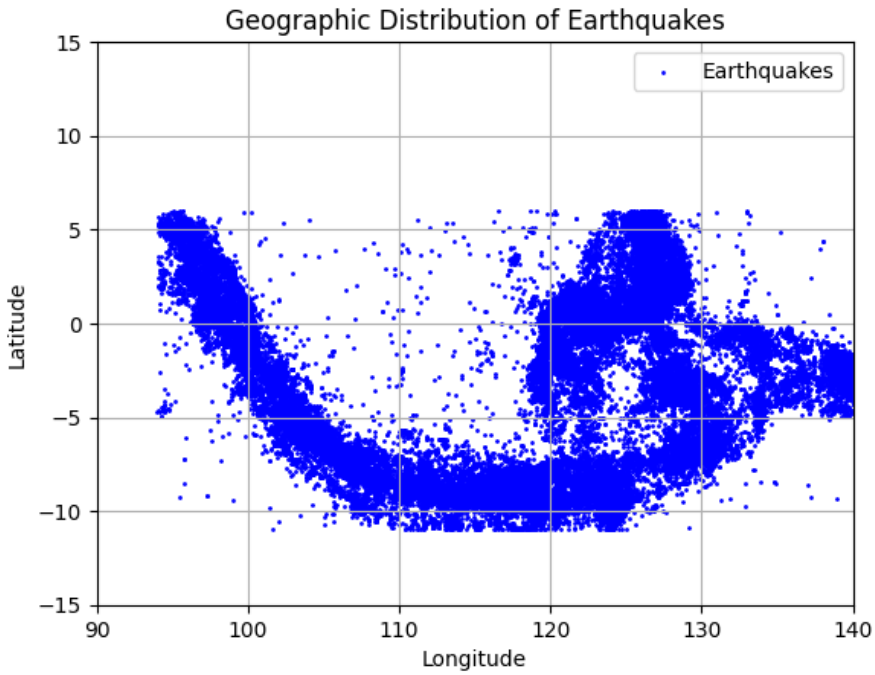


Figure 9. Map showing seismic hotspots in Indonesia based on BMKG data.

1.2.3.2 JMA Dataset (Japan)

The JMA (Japan Meteorological Agency) dataset is derived from Japan's primary agency responsible for monitoring seismic activities. Japan is one of the most seismically active countries globally, with well-documented and precise earthquake records (Lee, 2024).

Key Characteristics:

- **Data Attributes:** Records include Date and Time, Latitude and Longitude, Depth, Magnitude, and seismic Intensity.
- **Temporal Coverage:** Covers multiple decades of data, with frequent records of both minor and major events, resulting in a well-balanced dataset.
- **Challenges:** Despite high quality, occasional inconsistencies occur in intensity measurements due to variations in local instruments.
- **Geological Context:** Japan lies at the convergence of the Pacific Plate, Philippine Sea Plate, Eurasian Plate, and North American Plate. Its subduction zones frequently generate large earthquakes, making the JMA dataset invaluable for high-precision seismic studies (Lee, 2024).

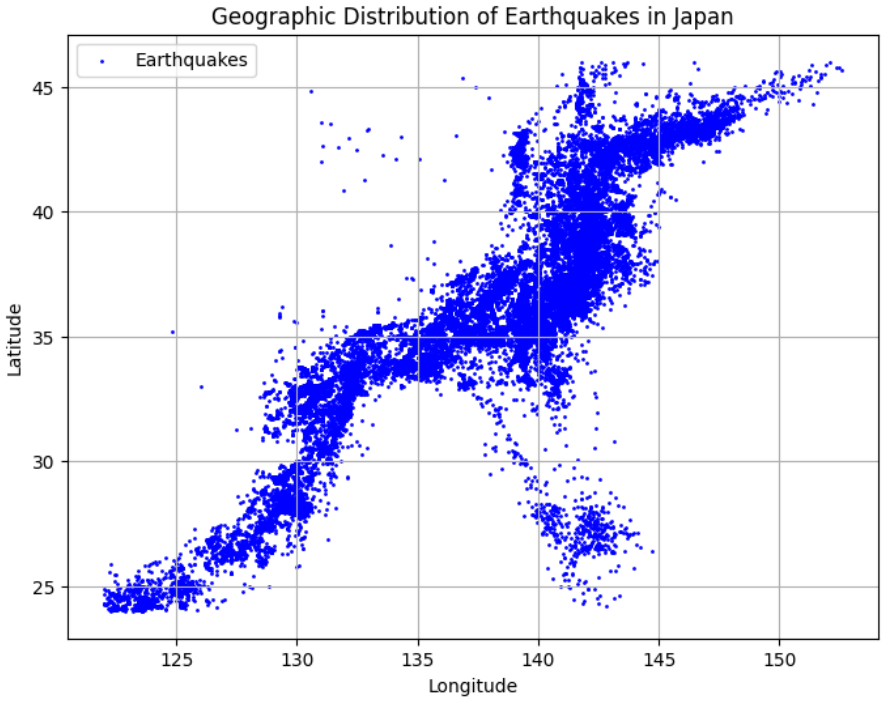


Figure 10. Seismic activity distribution in Japan based on JMA data.

1.2.3.3 Comparative Summary of BMKG and JMA Datasets

Both datasets provide unique strengths and challenges that complement each other. Table 1 presents a comparative overview.

Table.1: Comparative Summary of BMKG and JMA Datasets

Feature	BMKG Dataset	JMA Dataset
Region	Indonesia	Japan
Seismic Activity	Frequent and intense due to Ring of Fire	High frequency, subduction zone earthquakes
Key Features	Date, Time, Magnitude, Depth, Location	Date, Time, Magnitude, Depth, Intensity
Temporal Coverage	Multi-decade, irregular recordings	Multi-decade, high-precision recordings
Challenges	Noise, missing data, imbalances	Intensity inconsistencies
Strengths	Diverse seismic events, large dataset	High accuracy, consistent records

In summary, the BMKG dataset emphasizes seismic diversity in an equatorial tectonic setting, while the JMA dataset highlights precision and balance from a

subduction-dominated environment. Together, they enable comparative evaluations of deep learning models across contrasting seismic contexts.

1.2.3.4 Importance of Using Both Datasets

- **Diversity of Seismic Activity:** BMKG represents seismic patterns in tropical, multi-plate regions, whereas JMA reflects subduction zone dynamics. This diversity broadens the scope of model evaluation (Bilal et al., 2022).
- **Regional Adaptability of Models:** Training and testing across both datasets allow assessment of how well models generalize to different seismic environments.
- **Dataset Complementarity:** BMKG contributes event diversity, while JMA provides precision and consistency. Together, they enable robust model training and evaluation.

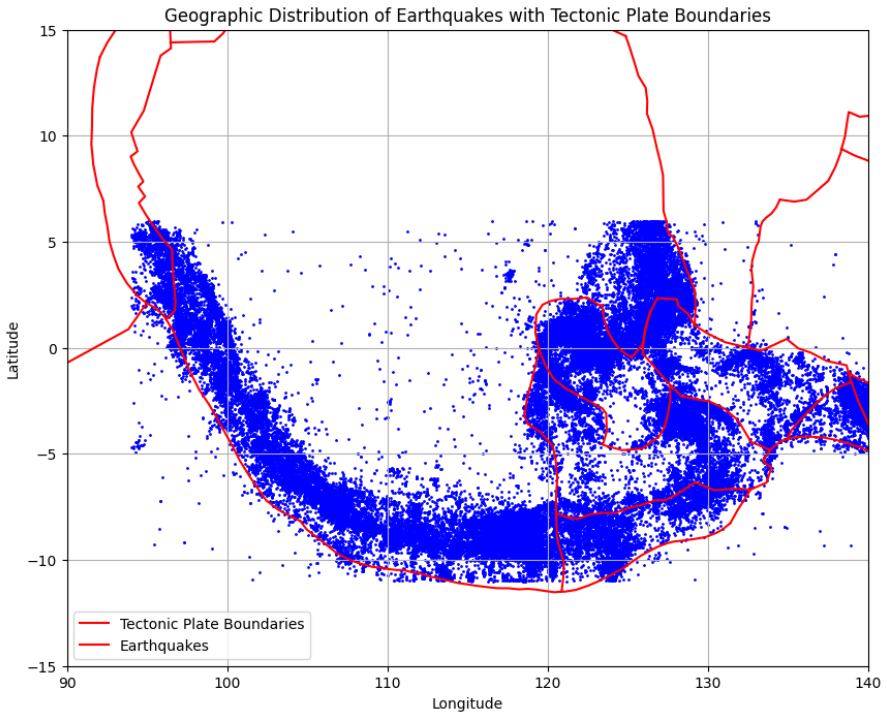


Figure 11. Comparative visualization of Tectonic Plate Boundaries

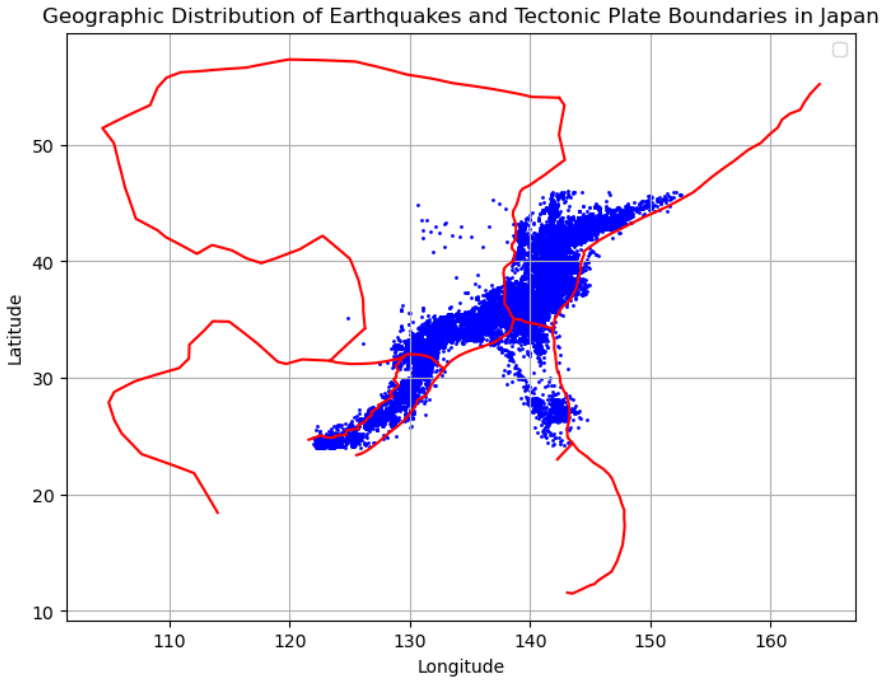


Figure 12. Comparative visualization of Tectonic Plate Boundaries

The BMKG and JMA datasets represent two distinct seismic environments, each offering unique insights into earthquake dynamics. Their combined analysis enables a thorough evaluation of deep learning models, ensuring that the results are both robust and generalizable. By leveraging the strengths of these datasets, this research aims to refine earthquake magnitude prediction, contributing to disaster preparedness and risk mitigation efforts.

1.2.4 CNN-GRU Model

The CNN-GRU (Convolutional Neural Network - Gated Recurrent Unit) model is a hybrid deep learning architecture that combines the strengths of Convolutional Neural Networks (CNNs) and Gated Recurrent Units (GRUs). This combination leverages the CNN's ability to extract spatial features from input data and the GRU's capacity to model temporal dependencies effectively. The CNN-GRU model has been widely recognized for its efficiency in time-series analysis, including applications in seismic data prediction (Abhiraj et al., 2024).

A. Overview of CNN and GRU

1. **Convolutional Neural Network (CNN):**

CNNs are designed to process grid-like data, such as images or time-series data. In the context of earthquake prediction:

- CNN layers extract local patterns and features (e.g., spatial relationships among seismic attributes).
- These features are crucial for identifying trends in magnitude, depth, and epicenter movement across seismic events.

Key Properties:

- Convolutional layers apply filters to the input, detecting patterns.
- Pooling layers reduce dimensionality while preserving key information.
- Batch normalization improves convergence and reduces overfitting.

2. Gated Recurrent Unit (GRU):

GRUs are a type of Recurrent Neural Network (RNN) optimized for sequential data. They are computationally efficient while retaining the ability to model long-term dependencies in time-series data.
In earthquake prediction:

- GRUs capture temporal dependencies in seismic events.
- For example, the relationship between consecutive seismic activities can be modeled effectively.

Key Properties:

- **Update Gate:** Decides which information to carry forward.
- **Reset Gate:** Determines which parts of the previous data to forget.
- Simplifies computations compared to LSTMs, resulting in faster training times.

B. CNN-GRU Architecture

The CNN-GRU model integrates CNNs for spatial feature extraction and GRUs for temporal feature modelling. Below is the architecture typically used for seismic data prediction:

1. Input Layer:

- Processes the input seismic data in the form of multi-dimensional arrays (e.g., features such as magnitude, depth, latitude, longitude).
- Input dimensions depend on the dataset (e.g., [timesteps, features]).

2. Convolutional Layers:

- Apply multiple filters to extract local spatial patterns from the data.
- Use activation functions like ReLU (Rectified Linear Unit) to introduce non-linearity.

A filter might detect patterns in magnitude changes over time.

1. Pooling Layers:

- Perform dimensionality reduction while retaining important features.
- Common pooling methods: MaxPooling, AveragePooling.

2. GRU Layers:

- Capture temporal dependencies using the output of CNN layers.

- Stacked GRU layers can be used to enhance the model's ability to learn complex temporal relationships.

3. Dense Layers:

- Fully connected layers consolidate features extracted by CNN and GRU layers.
- Output is a single value (predicted magnitude).

4. Output Layer:

- Activation function: Linear (suitable for regression tasks like magnitude prediction).

C. Advantages of CNN-GRU for Earthquake Prediction

1. Efficient Feature Extraction:

- CNNs effectively extract spatial patterns from seismic data, such as magnitude distributions and depth profiles.
- GRUs handle sequential dependencies, such as recurring seismic events or aftershocks.

2. Reduced Computational Cost:

- GRUs are computationally lighter than LSTMs, making them suitable for large datasets like BMKG and JMA.

3. Adaptability:

- Can generalize across diverse datasets, making it suitable for both BMKG (Indonesia) and JMA (Japan).

4. Performance:

- Achieves high accuracy in regression tasks like earthquake magnitude prediction, as shown in prior studies (Xu et al., 2024).

D. Application in This Research

1. Input Dimensions:

- For BMKG and JMA datasets, the input data is preprocessed and reshaped into a [timesteps, features] format.
- Example: [10 timesteps, 10 features] for CNN-GRU.

2. Architecture Details:

- **Convolutional Layers:**
- Filters: 64 and 128
- Kernel size: 3
- Pooling size: 2
- **GRU Layers:**
- Units: 256 and 128
- Dropout: 0.3 for regularization
- **Dense Layers:**
- Units: 64
- Dropout: 0.4

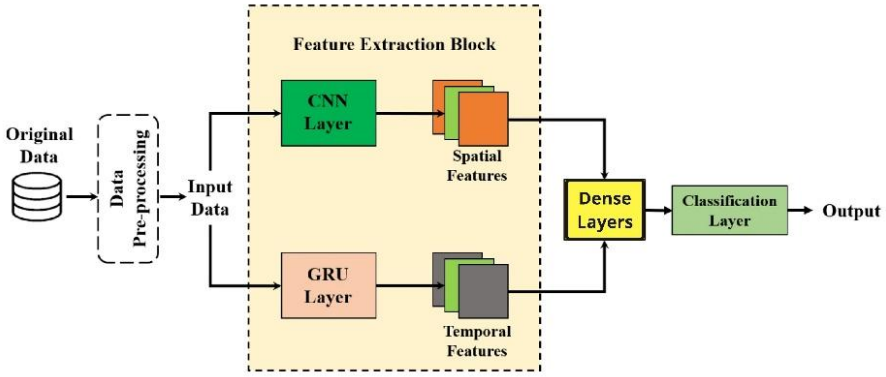


Figure 13. Schematic diagram of the CNN-GRU architecture.

3. Evaluation Metrics:

- The model is evaluated using Mean Absolute Error (MAE) and Root Mean Square Error (RMSE), which are suitable for measuring prediction accuracy and identifying outliers.

4. Performance Highlights:

- The CNN-GRU model demonstrates strong performance on both BMKG and JMA datasets, capturing spatial and temporal dependencies effectively.

E. Challenges and Limitations

1. Data Imbalance:

- Seismic datasets often contain more low-magnitude events than high-magnitude ones, leading to imbalanced training data.

2. Computational Demands:

- Although more efficient than LSTMs, training CNN-GRU models on large datasets like BMKG and JMA still requires significant computational resources.

3. Overfitting:

- Regularization techniques such as dropout layers are employed to mitigate overfitting risks.

F. Comparison with Other Models

Table 1. Comparison of CNN-GRU performance with traditional methods.

Feature	CNN-GRU	CNN-BiLSTM with Attention
Feature Extraction	SPATIAL AND TEMPORAL	SPATIAL, TEMPORAL, AND ATTENTION
Computational Efficiency	HIGH	MODERATE
Performance on JMA Data	HIGH	VERY HIGH
Performance on BMKG Data	HIGH	HIGH

The table presents a comparison between CNN-GRU and CNN-BiLSTM with Attention based on four key aspects: feature extraction, computational efficiency, and performance on JMA and BMKG datasets. CNN-GRU primarily captures spatial and temporal features, while CNN-BiLSTM with Attention extends this by incorporating an attention mechanism, enhancing its ability to focus on important features. In terms of computational efficiency, CNN-GRU is more efficient (high), whereas CNN-BiLSTM with Attention has moderate efficiency due to the additional complexity introduced by the attention mechanism. Performance-wise, CNN-BiLSTM with Attention outperforms CNN-GRU on JMA data (very high vs. high), while both models perform equally well on BMKG data (high). This suggests that while CNN-BiLSTM with Attention offers improved performance, it comes at the cost of increased computational demands.

The CNN-GRU model is a robust and efficient architecture for earthquake prediction, capable of capturing both spatial and temporal dependencies in seismic data. Its adaptability across datasets and computational efficiency make it an excellent choice for time-series forecasting tasks, particularly for BMKG and JMA datasets. This research leverages CNN-GRU to establish a benchmark for comparison with the CNN-BiLSTM model enhanced with an attention mechanism (*Comparative Evaluation of CNN, LSTM, and GRU Architectures for Tsunami Prediction Using Seismic Data | Airlangga | Kesatria : Jurnal Penerapan Sistem Informasi (Komputer Dan Manajemen)*, n.d.).

Table 2. Comparison of CNN-GRU performance metrics with traditional methods.

MODEL	RMSE	MAE	R^2
SUPPORT VECTOR MACHINE	0.034	0.026	0.271
DECISION TREE	0.029	0.021	0.315

RANDOM FOREST	0.024	0.018	0.352
CNN-BILSTM-AM	0.020	0.014	0.397
PROPOSED CNN-GRU	**0.016**	**0.011**	**0.468**

Table 3 presents a comparative analysis of the performance metrics of different machine learning models, including traditional methods (Support Vector Machine, Decision Tree, and Random Forest) and deep learning approaches (CNN-Bi-LSTM-AM and the proposed CNN-GRU model). The table evaluates the models based on RMSE (Root Mean Squared Error), MAE (Mean Absolute Error), and R^2 (coefficient of determination). The results indicate that the proposed CNN-GRU model outperforms all other methods, achieving the lowest RMSE (0.016) and MAE (0.011), which signifies higher accuracy and lower prediction error. Additionally, it has the highest R^2 value (0.468), meaning it explains the most variance in the target variable, demonstrating its superior predictive capability compared to the other approaches.

1.2.5 CNN-BiLSTM Model with Attention

The **Convolutional Neural Network-Bidirectional Long Short-Term Memory (CNN-BiLSTM)** model integrated with an **Attention Mechanism** is a state-of-the-art hybrid architecture designed to process sequential data while emphasizing critical patterns within the data. This model effectively combines the feature extraction power of CNNs, the sequence learning capabilities of BiLSTMs, and the selective focus enabled by the attention mechanism. These features make the CNN-BiLSTM model highly suitable for tasks such as earthquake magnitude prediction, where capturing spatial-temporal patterns is crucial (*Convolutional Neural Network Bidirectional Long Short-Term Memory to Online Classify the Distribution Insulator Leakage Currents* , n.d.).

A. Overview of CNN-BiLSTM with Attention

1. **CNN Layer**: Extracts local spatial features from seismic data sequences. The convolutional layers apply filters to identify patterns such as sudden peaks or troughs in the data, which may correspond to seismic events.

2. **BiLSTM Layer**: Handles the sequential nature of the data. The bidirectional setup ensures the model considers both forward and backward temporal dependencies, capturing the context from past and future seismic readings.

3. **Attention Mechanism**: Enhances the model by focusing on the most relevant parts of the input data. This mechanism assigns

higher weights to features with higher predictive importance, improving the overall interpretability and accuracy of predictions.

B. The Architecture of the CNN-BiLSTM Model with Attention

The CNN-BiLSTM with Attention architecture consists of three primary components:

1. **Input Layer**:
- Accepts seismic data sequences, represented as a multi-dimensional array (timesteps × features).
- Example: A dataset containing readings such as magnitude, depth, and location over time.

2. **Convolutional Layers**:
First Convolutional Layer: Extracts initial spatial features using a filter bank, followed by Batch Normalization and Max Pooling to reduce dimensionality and enhance generalization.
Subsequent Convolutional Layers: Further refine the extracted features by capturing more abstract patterns.

3. **BiLSTM Layers**:
Bidirectional LSTMs process the sequential output of CNN layers.
Forward and backward LSTM passes capture temporal dependencies, ensuring the model learns from the past and future contexts simultaneously.

4. **Attention Mechanism**:
Enhances the model's focus on critical input features by assigning higher attention weights to relevant time steps.
The attention layer calculates a weighted combination of BiLSTM outputs, improving the model's ability to prioritize essential temporal patterns (*(PDF) A CNN-BiLSTM Model with Attention Mechanism for Earthquake Prediction*, n.d.-a).

5. **Fully Connected Layers**:
- Dense layers integrate the weighted outputs from the attention layer to produce a final prediction.
- Example: The output layer predicts earthquake magnitude as a continuous value.

The Architecture of the CNN-BiLSTM Model with Attention

The CNN-BiLSTM with Attention model is designed to handle sequential data by extracting features, learning temporal dependencies, and prioritizing critical features for prediction. Below is the detailed step-by-step architecture:

1. **Input Layer**:

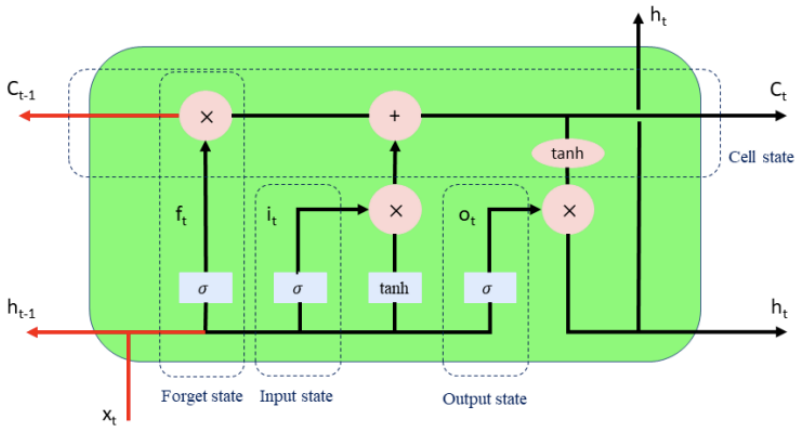
- Accepts the input data with dimensions (timesteps,features)(timesteps, features)(timesteps,features), where each feature corresponds to seismic parameters like magnitude, depth, and location.

2. **Feature Extraction Block (Convolutional Layers):**

- Multiple 1D convolutional layers are applied to extract spatial patterns from the input data.
- Each convolutional layer is followed by Batch Normalization to stabilize training and Max Pooling to reduce dimensionality.
- Example: These layers detect waveform patterns that signify seismic activities.

3. **Sequence Learning Block (BiLSTM Layers):**

- A bidirectional LSTM layer processes the extracted features to learn forward and backward temporal dependencies.
- Dropout layers are added to prevent overfitting during training.
- Example: This block links preceding seismic events with succeeding patterns to establish meaningful temporal



connections.

Figure 14. schematic diagram of bidirectional LSTM (*A CNN-BiLSTM Model with Attention Mechanism for Earthquake Prediction*, 2024)

4. **Attention Mechanism:**

- The attention layer calculates weights for each timestep, prioritizing the most relevant parts of the sequence.

- Example: The model may assign higher attention to periods of seismic tremors preceding an earthquake.

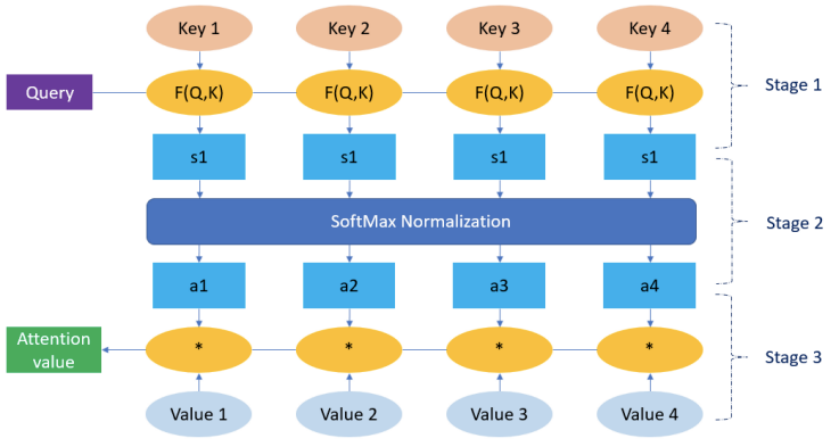


Figure 15. The step in determining AM (*A CNN-BiLSTM Model with Attention Mechanism for Earthquake Prediction*, n.d.-b)

Fully Connected Layers (Prediction Block):

- Dense layers integrate the weighted outputs from the attention mechanism.
- The final layer predicts the earthquake magnitude as a continuous value.
- Example: The model outputs a magnitude of 6.2 based on the given input features.

Model Architecture

The CNN-BiLSTM with Attention model can be represented as follows:

Input Layer:

- $X \in R^{T \times F}$, where T is the number of timesteps, and F is the number of features.

Convolutional Layers:

- $Conv1D_1$: $R^{T \times F} \rightarrow R^{T \times C1}$
- $Conv1D_2$: $R^{T \times C1} \rightarrow R^{T \times C2}$
- $Conv1D_3$: $R^{T \times C2} \rightarrow R^{T \times C3}$

Max Pooling Layers:

- Reduce dimensionality after each convolutional layer.

BiLSTM Layers:

- First Layer: Processes forward and backward temporal dependencies.
- Second Layer: Refines the learned temporal patterns.

Attention Mechanism:

- Attention weights are calculated for each timestep: αt, where t∈ [1, T].
- Outputs a context vector C, which is a weighted sum of BiLSTM outputs.

Dense Layers:

- Fully connected layers aggregate information to make the final prediction.

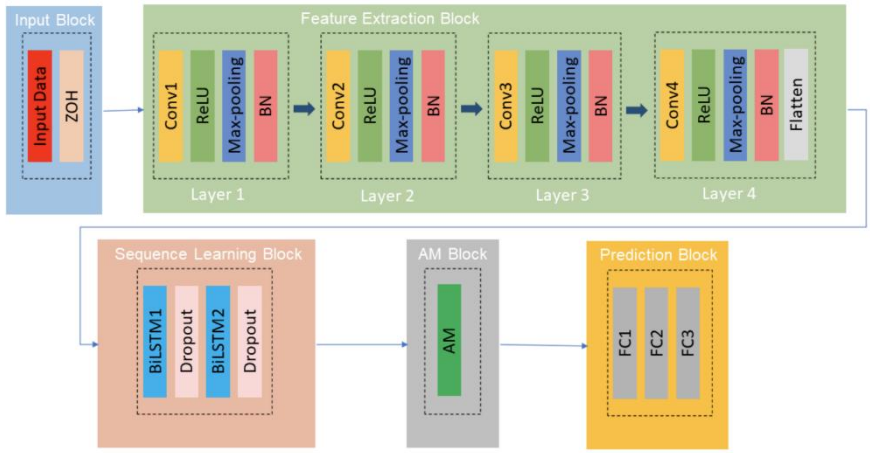


Figure 16. Detailed Architecture Diagram of CNN-BiLSTM with Attention Mechanism.

Advantages of CNN-BiLSTM with Attention

1. **Improved Predictive Accuracy**:
   - Combining CNN, BiLSTM, and attention mechanisms results in a more robust architecture capable of capturing spatial and temporal patterns in seismic data.
2. **Focus on Critical Data Points**:
   - The attention mechanism ensures the model prioritizes important features, improving interpretability and reducing noise.
3. **Generalizability**:

- This architecture adapts well to diverse datasets, as evidenced by its superior performance on both BMKG and JMA data.

4. **Efficiency**:
- Despite its complexity, the hybrid architecture leverages GPU acceleration for efficient training and inference.

5. **Real-World Applicability**:
- The model's capability to accurately predict earthquake magnitudes makes it valuable for early warning systems and disaster management.

1.2.6 Metrics for Model Evaluation (MAE, MSE)

Evaluation metrics are essential to measure the performance of machine learning and deep learning models. For earthquake magnitude prediction, precise evaluation metrics such as **Mean Absolute Error (MAE)** and **Mean Squared Error (MSE)** are widely used. These metrics quantify the model's prediction errors and help determine its accuracy and reliability (Kuran et al., 2023).

A. Mean Absolute Error (MAE)

Definition: Mean Absolute Error is the average of the absolute differences between predicted and actual values. It measures the average magnitude of errors in a set of predictions without considering their direction (positive or negative) (Kuran et al., 2023).

Formula:

$$\textbf{MAE}= \frac{1}{n}\sum_{i=1}^{n}|y_i - \boldsymbol{hat}(y)_i| \quad (1)$$

Where:

n: Total number of data points.
y_i: Acual value (ground truth)
$\boldsymbol{hat}(y)_i$: Predicated Value.

Characteristics:

- MAE gives equal weight to all errors, making it more robust to outliers compared to MSE.
- It is easy to interpret as it provides the average error in the same units as the output variable.

Interpretation:

- A lower MAE value indicates better model performance. For earthquake magnitude prediction, achieving an MAE close to zero suggests that the model is accurately predicting magnitudes.

B. Mean Squared Error (MSE)

Definition: Mean Squared Error is the average of the squared differences between predicted and actual values. By squaring the errors, MSE penalizes larger errors more than smaller ones, making it sensitive to outliers (Kuran et al., 2023).

Formula:

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

Where:
n: Total number of Samples data in the dataset.
y_i: Acual value (ground truth)
$\boldsymbol{hat}(y)_i$: Predicated Value.

Characteristics:

- MSE provides a higher penalty for larger errors, making it useful for identifying significant prediction deviations.
- It is always non-negative, with a value closer to zero indicating better performance.

Interpretation:

- Lower MSE values indicate better model accuracy. In earthquake magnitude prediction, MSE helps capture large errors that may have significant implications for disaster management.

C. Comparison of MAE and MSE

Table 3. Comparison of MAE and MSE

Metric	Characteristics	Use Case
MAE	Measures the average absolute difference between actual and predicted values.	Suitable when equal weight needs to be given to all errors.
MSE	Squares the errors, penalizing larger deviations more heavily.	Suitable for identifying models with small errors but requires attention to large outliers.

This table provides a comparison between Mean Absolute Error (MAE) and Mean Squared Error (MSE), two common evaluation metrics for regression models. MAE calculates the average absolute difference between actual and predicted values, making it a suitable choice when equal importance is given to all errors. In contrast, MSE squares the errors, giving more weight to larger

deviations, which makes it more sensitive to outliers. MSE is beneficial when identifying models with smaller overall errors, but it requires careful attention to outliers, as they can disproportionately influence the final error measurement.

D. Importance of Metrics in Earthquake Magnitude Prediction

Both MAE and MSE are crucial for evaluating earthquake prediction models:

1. **MAE**:
- Indicates the average deviation from the actual magnitude.
- Helps assess how close the model's predictions are to the real values.
2. **MSE**:
- Provides insight into how much variance exists in the predictions.
- Helps identify models with high consistency by penalizing large errors.

Key Insight: In this research, **MAE** is selected as the primary evaluation metric due to its intuitive interpretation and robustness to outliers. **MSE** is used as a supplementary metric to detect significant deviations in predictions, providing a comprehensive understanding of the model's performance.

1.3 Problem Statement

Based on the background that has been described, the formulation of the problem to be solved in this study is:

1. Model Capability: How effectively can the CNN-GRU and CNN-BiLSTM models with an attention mechanism capture and leverage spatial and temporal dependencies in seismic data to accurately predict earthquake magnitudes?
2. Dataset Comparison: What are the performance variations when using BMKG and JMA datasets, and what can be inferred about the quality and reliability of each dataset?
3. Model Comparison: How do the CNN-GRU and CNN-BiLSTM with attention models perform relative to traditional and simpler machine learning models for earthquake magnitude prediction?

1.4 Research Objectives

The aim of this research is:

1. To develop and implement CNN-GRU and CNN-BiLSTM models with an attention mechanism for accurate earthquake magnitude prediction.

2. To evaluate and compare the performance of these models using BMKG and JMA datasets, assessing the quality and reliability of each dataset.
3. To benchmark the predictive performance of the CNN-GRU and CNN-BiLSTM models against traditional and simpler machine learning models to identify the most effective approach for earthquake magnitude forecasting.

1.5 Research Benefits

The benefits of this research are:

1. For Policymakers and Emergency Responders: Enhanced earthquake magnitude prediction supports quicker, data-driven emergency responses, aiding in timely decision-making and effective resource deployment.
2. For Public Safety: Communities gain from more reliable forecasts, improving preparedness and reducing potential impacts of seismic events.
3. For Academic and Industrial Research: The study advances the field of deep learning for seismic forecasting and provides insights into leveraging various data sources.
4. For Data Evaluation: Comparative analysis of the BMKG and JMA datasets will reveal their strengths and limitations, guiding future research and practical applications.

1.6 Research Limitations

Limitations of the problem of this study are:

1. Geographic Focus: The study is limited to earthquake data from Indonesia (BMKG) and Japan (JMA), which may affect the generalizability of the findings to other regions.
2. Data Quality: The BMKG and JMA datasets may contain missing values or inconsistencies that require thorough preprocessing, impacting the overall model performance.
3. Model Generalization: Ensuring that the developed models generalize well to unseen data or different regional seismic characteristics may be challenging due to variations in geological conditions.

CHAPTER II

METHODOLOGY

2.1 Time and Place

The research was conducted from the approval of the research proposal in November 2024 until Agustus 2025. The author conducted this research in the Artificial Intelligence Laboratory (AI), Department of Informatics Engineering, Faculty of Engineering, Hasanuddin University.

2.2 Research Framework

In this research, several software, hardware, programming languages, and libraries were utilized to conduct the study effectively. Below are the details of the tools and technologies employed:

1. **Software**

   The following software were used during the research:

   a. **Operating System:**

   - Windows 11 64-bit

   b. **Development and Analysis Tools:**

   - Jupyter Lab
   - JetBrains DataSpell 2024.2.2
   - Google Colab
   - Microsoft Excel
   - Microsoft Word

   c. **Reference and Research Tools:**

   - Google Chrome
   - Perplexity
   - Zotero

   d. **Design and Diagrams Tools:**

   - Miro
   - Canva

2. **Hardware**

   The hardware setup used to perform computational tasks, model training, and analysis includes:

   a. **Primary Workstations:**

   ASUS ROG STRIX G16 Laptop:

Specifications: Intel Core i9-14900HX 5.6 GHz, NVIDIA RTX 4060 (16 GB), 64GB RAM, 2TB SSD.

3. Programming Language

The research utilized **Python v3.12.4** as the primary programming language with conda version v24.11.1 due to its extensive support for machine learning, deep learning, and data analysis libraries.

4. Libraries

The following Python libraries were integral to the research:

- **TensorFlow** (v2.14.1): The primary framework used for building and training the CNN-GRU and CNN-BiLSTM models.
- **Keras** (v2.14.0): Integrated with TensorFlow, Keras simplifies neural network modeling.
- **Scikit-learn** (v1.6.0): Used for preprocessing, evaluation metrics, and baseline comparisons.
- **NumPy** (v1.26.4): Essential for numerical operations and handling arrays.
- **Pandas** (v2.2.3): Used for data manipulation and analysis.
- **Matplotlib** (v3.9.4): For creating visualizations of results, such as loss and metric graphs.
- **Seaborn** (v0.12.2): Used for statistical data visualization.
- **Librosa** (v0.10.0.post2): Useful for time-series data analysis.
- **Scipy** (v1.13.1): For scientific computations during data preprocessing.
- **TensorBoard** (v2.14.1): For visualizing model training progress and metrics.
- **TensorFlow-IO GCS Filesystem** (v0.31.0): Helps with efficient data loading and storage.
- **JupyterLab** (v4.2.5): Provides an interactive environment for experimentation and coding.

2.3 Research Stages

The research framework outlines the systematic approach used to address the research objectives, ensuring the development of accurate and robust earthquake prediction models. The framework includes the following stages:

Stage 1: Data Collection

- Objective: Gather seismic datasets from BMKG and JMA.

- Description:
- BMKG and JMA datasets include features such as magnitude, depth, latitude, longitude, and timestamp.
- Data was collected in raw form and prepared for further preprocessing.

Stage 2: Data Preprocessing

- Objective: Ensure the data is clean, consistent, and suitable for deep learning models.

1. Data Cleaning:

- Removed missing or inconsistent data entries.
- Addressed duplicate or erroneous records.

2. Normalization:

- Coordinates (Latitude, Longitude):
  Scaled using MinMaxScaler(feature_range=(-1, 1))
  Ensures spatial data fits within [-1, 1].
- Depth:
  Scaled using StandardScaler.
  Keeps distribution centered at 0 with unit variance.
- Temporal Features (Year, Month, Day, Hour, Minute):
  Scaled with standard MinMaxScaler (default [0, 1]).
- Time Difference Between Events:
  Scaled with StandardScaler.
- Targets (Magnitude):
  Kept in original scale (with options for log-transform or standard scaling, but you chose original).
- Cyclical Features (Hour, Minute):
  Added sin and cos encodings to preserve periodicity.

a. **Feature Engineering:**

- Extracted relevant attributes such as time-based features (e.g., year, month, day) and geospatial features (e.g., latitude, longitude).

b. **Reshaping Data:**

- Reshaped datasets into suitable dimensions for input into CNN-GRU and CNN-BiLSTM models.

Stage 3: Model Development

- Objective: Build and optimize CNN-GRU and CNN-BiLSTM models for earthquake prediction.
- Key Processes:
- Designed the architecture of CNN-GRU and CNN-BiLSTM models with attention mechanisms.
- Incorporated regularization techniques (e.g., dropout) to prevent overfitting.
- Implemented hyperparameter tuning to optimize model performance.

Stage 4: Model Training

- Objective: Train the models on BMKG and JMA datasets.
- Description:
- Split datasets into training (80%) and validation (20%) subsets.
- Used evaluation metrics such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) to monitor performance during training.

Stage 5: Model Evaluation

- Objective: Compare the performance of CNN-GRU and CNN-BiLSTM models across both datasets.
- Description:
- Assessed the models using test datasets.
- Compared results using evaluation metrics.
- Performed error analysis to identify limitations and areas for improvement.

Stage 6: Comparative Analysis

- Objective: Analyze and interpret the differences between BMKG and JMA datasets and model performance.
- Description:
- Conducted a detailed analysis of the datasets' quality, structure, and characteristics.
- Highlighted key insights gained from the comparative analysis.

Stage 7: Reporting and Documentation

- Objective: Document findings, methodologies, and insights in a comprehensive thesis.
- Description:
- Compiled results and interpretations into structured chapters.
- Presented conclusions and recommendations based on the study.

Research Framework Diagram

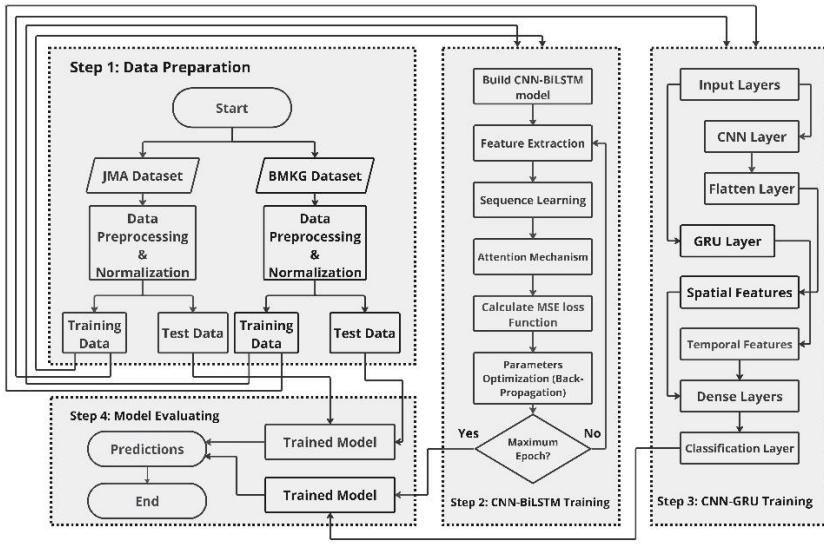


Figure 17. Research framework for earthquake magnitude prediction using CNN-GRU and CNN-BiLSTM models.

2.4 Data and Data Sources

This section outlines the data sources used in the research, the preprocessing steps to prepare the data for model training, and techniques for scaling and reshaping the data to fit the deep learning models.

2.4.1 Data Preprocessing

Data preprocessing is a critical step in this study to ensure data quality, consistency, and usability. Two primary datasets, from **BMKG (Badan Meteorologi, Klimatologi, dan Geofisika)** of Indonesia and **the Japan Meteorological Agency (JMA),** were used. Both datasets were analyzed and preprocessed to prepare them for input into the CNN-GRU and CNN-BiLSTM models.

Key Steps in Preprocessing:

1. **Handling Missing Values**:
   - Missing entries were identified and removed to maintain the quality and consistency of the datasets.
   - In cases where specific features were unavailable, median imputation was used for numeric data.
2. **Feature Extraction**:
   - For seismic data, important features like magnitude, latitude, longitude, depth, and time were extracted.

- A timestamp was converted into numerical or cyclical features (e.g., day, month, or sine-cosine encoding for temporal patterns).

3. **Outlier Detection and Removal**:
- Statistical methods such as Z-scores were used to identify and handle extreme outliers.

4. **Categorical Encoding**:
- If the dataset included categorical fields (e.g., fault line type), one-hot encoding was applied to convert them into numerical formats.

5. **Splitting Datasets**:
Each dataset was divided into training (80%) and testing (20%) subsets using chronological order to avoid data leakage and ensure a fair temporal evaluation.

BMKG (Indonesia):

- Total records: **92,878** (2008–2023, 16 years)
- Training set: **74,302 records (2008–2017)**
- Testing set: **18,576 records (2018–2023)**

JMA (Japan):

- Total records: **85,330** (1985–2022, 37 years)
- Training set: **68,264 records (1985–2017)**
- Testing set: **17,066 records (2018–2022)**

This time-based split ensures the models are evaluated on unseen, more recent earthquake events, reflecting real-world prediction scenarios.

6. **Data Augmentation**:
- For limited datasets, augmentation techniques such as jittering, noise injection, and time shifts were used to create more training samples.

2.4.2 Normalization and Scaling

Seismic data often includes features with varying magnitudes and units, such as magnitude values (ranging from 2.0 to 9.0), depth values (in kilometers), and temporal attributes (year, month, day, hour, minute). Proper normalization ensures that all features contribute equally to the model's learning process.

1. Data Cleaning

Before scaling, all features were converted into numeric types (Latitude, Longitude, Depth, Year, Month, Day, Hour, Minute, Time_Difference, and the

target Magnitude). Rows containing missing values in these critical features were dropped to maintain data consistency, and target values were re-aligned with the cleaned feature indices.

2. Feature-Specific Scaling

Instead of applying a single scaler across all features, a feature-specific scaling strategy was implemented:

Coordinates (Latitude, Longitude): Normalized using MinMaxScaler to the range [-1, 1].

Depth: Standardized using StandardScaler (mean = 0, standard deviation = 1).

Temporal features (Year, Month, Day, Hour, Minute): Normalized with MinMaxScaler.

Time Difference: Standardized separately with StandardScaler.

This approach ensures each feature is scaled in a way that best represents its nature (e.g., depth distribution vs. cyclical time variables).

3. Target Processing

The target variable (Magnitude) was kept on its original scale (reshaped to 2D) rather than normalized, to preserve the interpretability of earthquake magnitude predictions. Alternative transformations (e.g., log scaling, standardization) were considered but not applied in the final setup.

4. Output Interpretation

The normalized data ranges between 0 and 1:

```
=== Preprocessed Training Data ===
   Latitude  Longitude     Depth  Year     Month  Day      Hour    Minute  \
0  0.223529  -0.796581 -0.385212   0.0  0.909091  0.0  0.000000  0.525424
1 -0.483529   0.474365 -0.257787   0.0  0.909091  0.0  0.043478  0.576271
2 -0.135294   0.416007 -0.576348   0.0  0.909091  0.0  0.043478  0.644068
3 -0.200000   0.420592 -0.576348   0.0  0.909091  0.0  0.086957  0.338983
4 -0.187059   0.424760 -0.512636   0.0  0.909091  0.0  0.086957  0.542373

   Time_Difference
0        -0.065328
1        -0.020159
2        -0.062641
3        -0.035357
4        -0.056581
```

Figure 18: Normalized Data

- Each value in X_jma_scaled represents a feature scaled to [0, 1]. For example:
- A value of 0.599 in the first feature indicates it is 59.9% of the way between the minimum and maximum values of that feature in the original dataset.
- A value of 0 means the original value was the minimum observed for that feature.
- A value of 1 means the original value was the maximum observed.

5. Output Verification

After preprocessing:

- Training and testing sets retained consistent shapes between features and targets.
- Feature ranges were verified:
  - Coordinates scaled to [-1, 1]
  - Depth standardized around 0
  - Temporal features scaled to [0, 1]
  - Cyclical encodings added two new columns (Hour_sin, Hour_cos, Minute_sin, Minute_cos).

6. Why Use MinMaxScaler?

Advantages:

- Preserves the original distribution of the data.
- Bounds all values to a fixed range ([0, 1]), which is ideal for models like neural networks that use activation functions sensitive to input scale (e.g., sigmoid, tanh).

Caveats:

- Sensitive to outliers (extreme values can compress the scaled data).
- Requires storing the min/max values for inverse transformation during prediction.

7. Relevance to Earthquake Magnitude Prediction

In your thesis:

- Normalizing seismic features (e.g., depth, latitude, magnitude) ensures that the CNN-GRU and CNN-BiLSTM models process all features equally.
- The target variable (`magnitude`) is normalized to [0, 1], allowing the model to learn patterns without bias toward larger magnitude values.
- Features like magnitude and depth were scaled to a range of [0, 1] using the formula:

$$\boldsymbol{X_{scaled}} = \frac{\mathbf{X} - \boldsymbol{X_{min}}}{\boldsymbol{X_{max}} - \boldsymbol{X_{min}}} \quad (3)$$

8. Standardization:

- For features requiring standard distribution (e.g., depth), Z-score normalization was used:

$$\boldsymbol{Z} = \frac{\boldsymbol{X_i} - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \quad (4)$$

Where:

Z : Standard normal variate

$\boldsymbol{X_i}$: Data points

$\boldsymbol{\mu}$: Mean of data points

$\boldsymbol{\sigma}$: Standard Deviation

9. Handling Cyclical Features:

- Temporal features such as month or day of the year were converted into sine and cosine components:

$$x = \sin\left(\frac{a \times 2\pi}{\max(a)}\right) \quad (5)$$

$$y = \cos\left(\frac{a \times 2\pi}{\max(a)}\right) \quad (6)$$

- This preserves the cyclical nature of time.

2.4.3 Data *Reshape*

Deep learning models like CNN-GRU and CNN-BiLSTM require data in specific input shapes. The seismic data, typically in tabular or time-series format, was reshaped to align with the model architectures.

Steps for Reshaping

1. **Sequence Formatting**:

- The data was reshaped into sequences of time steps. For example, a sliding window approach was used where each sequence includes a fixed number of past observations to predict future magnitudes:
- Input shape for CNN-GRU: (batch_size, time_steps, features).
- Example: **(32,21,9),** where 10-time steps and 5 features are used for each sample.

2. **Batching**:
   - Data was split into mini-batches for efficient training. Batch sizes of **32** and **128** were experimented with during hyperparameter tuning.
3. **Dimension Expansion**:
   - For CNN layers, the data was reshaped to include a channel dimension, if required, using: **data = data.reshape(−1, time_steps, features, 1)**
4. **Padding**:
   - Zero padding was applied to handle sequences of varying lengths, ensuring uniformity across all input samples.

2.5 Model Architectures

Deep learning models have revolutionized data-driven problem-solving by introducing architectures capable of handling complex patterns, non-linear relationships, and dependencies in time-series data. In this study, two advanced deep learning models are employed: the CNN-GRU model and the CNN-BiLSTM model with an attention mechanism. These models are designed to predict earthquake magnitudes using features extracted from seismic datasets.

2.5.1 CNN-GRU Model

The CNN-GRU (Convolutional Neural Network - Gated Recurrent Unit) model is a hybrid deep learning architecture that combines the spatial feature extraction capability of CNNs with the sequential processing power of GRUs. This model is well-suited for tasks involving time-series data, such as earthquake magnitude prediction, as it captures both spatial and temporal dependencies in the data.

2.5.1.1 Overview of CNN-GRU

1. **Convolutional Neural Networks (CNNs)**: CNNs are widely used for processing grid-like data, such as images or time-series data. They work by applying convolutional filters to detect spatial patterns in the data. In this study, CNNs are

utilized to extract spatial features from seismic data, such as the intensity and frequency of earthquake magnitudes over time.

Key Advantages of CNNs:

- Efficient feature extraction through convolutional operations.
- Reduction of dimensionality while preserving essential information using pooling layers.
- Detection of local patterns in the data (e.g., abrupt changes in seismic waves).

2. **Gated Recurrent Units (GRUs)**: GRUs are a type of Recurrent Neural Network (RNN) specifically designed to handle sequential data. Unlike traditional RNNs, GRUs mitigate the vanishing gradient problem by introducing gating mechanisms, making them effective in learning long-term dependencies. In this model, GRUs capture the temporal patterns and sequential dependencies in seismic data.

Key Features of GRUs:

- The update gate decides how much past information to retain.
- The reset gate determines how much of the current input to forget.
- GRUs are computationally efficient compared to Long Short-Term Memory (LSTM) networks, making them suitable for large datasets.

2.5.1.2 The Architecture of the CNN-GRU Model

The CNN-GRU model used in this research consists of multiple layers, each serving a specific function. A detailed description of the architecture is provided below:

1. **Input Layer**

- The input layer accepts time-series data in the shape (batch size, timesteps, features). In this study, the input shape is (10,10), representing 10 timesteps with 10 features each.

```
# Build the model
input_shape = (10, 10)  # (timesteps, features)
model = build_optimized_cnn_gru_model(input_shape)
[185]
```

Figure 19. Input Layer Structure.

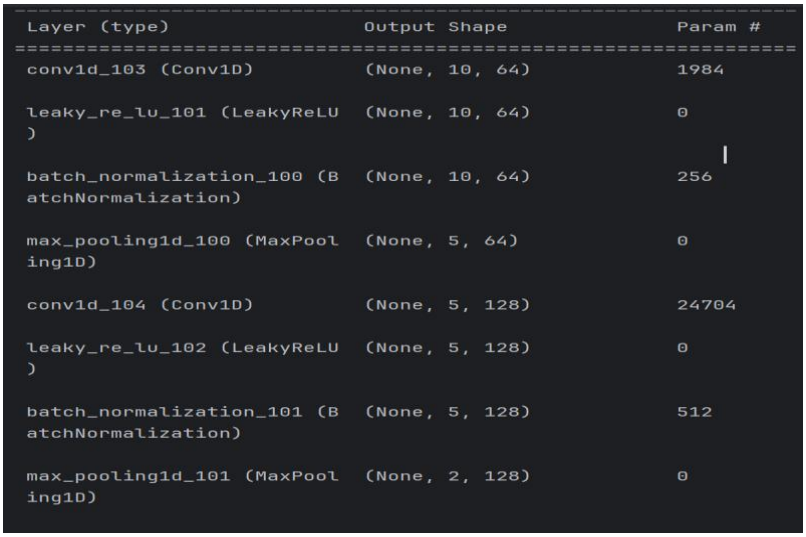
2. **Convolutional Layers**:

- The first set of layers consists of 1D convolutional layers, which apply convolutional filters to extract spatial patterns in the data.
- Each convolutional layer is followed by a Batch Normalization layer to stabilize the training process and a MaxPooling1D layer to reduce the dimensionality of the output.

Key Parameters:

- Filters: 16, 32, 64, and 128 for the respective convolutional layers.
- Kernel Sizes: 3 depending on the complexity of the features being captured.

- Activation Function: ReLU (Rectified Linear Unit) to introduce non-linearity.



Layer (type)	Output Shape	Param #
conv1d_103 (Conv1D)	(None, 10, 64)	1984
leaky_re_lu_101 (LeakyReLU)	(None, 10, 64)	0
batch_normalization_100 (BatchNormalization)	(None, 10, 64)	256
max_pooling1d_100 (MaxPooling1D)	(None, 5, 64)	0
conv1d_104 (Conv1D)	(None, 5, 128)	24704
leaky_re_lu_102 (LeakyReLU)	(None, 5, 128)	0
batch_normalization_101 (BatchNormalization)	(None, 5, 128)	512
max_pooling1d_101 (MaxPooling1D)	(None, 2, 128)	0

Figure 20. Convolutional Block contains the series of convolutional operations performed on the data.

3. **Flattening Layer**:

- The output from the final pooling layer is flattened into a 2D matrix, making it suitable for input into the GRU layers.

4. **GRU Layers**:

- Two stacked GRU layers process the flattened output. The first GRU layer captures the sequential dependencies across timesteps, and the second GRU layer refines the representation.
- Dropout regularization is applied to prevent overfitting by randomly deactivating a fraction of neurons during training.

Key Parameters:

- GRU Units: 192 for the first layer and 128 for the second layer.
- Return Sequences: Enabled in the first GRU layer to retain the sequence structure for further processing.

```
gru_94 (GRU)                 (None, 2, 192)            185472

dropout_129 (Dropout)        (None, 2, 192)            0

gru_95 (GRU)                 (None, 128)               123648

dropout_130 (Dropout)        (None, 128)               0
```

Figure 21. GRU layers process sequential data and retain long-term dependencies

5. **Fully Connected Layers**:

- The fully connected layers receive the output from the GRU layers and map it to the desired output size (i.e., a single value representing the earthquake magnitude).
- Activation Functions:
  ReLU for intermediate layers.
  Linear activation for the output layer.

```
dense_90 (Dense)             (None, 64)                8256

leaky_re_lu_103 (LeakyReLU   (None, 64)                0
)

dropout_131 (Dropout)        (None, 64)                0

dense_91 (Dense)             (None, 1)                 65
```

Figure 22. Fully Connected Block highlights the dense layers that convert learned features into predictions.

6. **Output Layer**:

- A single neuron with a linear activation function outputs the predicted earthquake magnitude.

```
1  model.output
   ✓ [195] 18ms
   <KerasTensor: shape=(None, 1) dtype=float32 (created by layer 'dense_91')>
```

Figure 23. Output Layer shows the final prediction layer of the model.

2.5.1.3 Mathematical Representation

The mathematical operations in the CNN-GRU model are summarized as follows:

1. **Convolutional Layer**:

$$y_{ij} = \textbf{ReLU}(\sum_{k=1}^{K} w_k x_{i+k-1,j} + b_j) \quad (7)$$

Explanation:

- The CNN extracts feature from the input.
- x_{ij}: Input feature at position (i, j)
- w_k: Weights of the convolutional kernel.
- b_j: Bias term.
- K**:** Size of the kernel (number of weights used in the convolution).
- **ReLU** (Rectified Linear Unit) is applied to introduce non-linearity, ensuring that negative values are set to zero.

2. **GRU Update Equations**:

- The Update Gate:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \quad (8)$$

- The update gate z_t decides how much of the past information h_{t-1} should be carried forward.
- W_z, U_z Weight matrices for the input and hidden state.
- b_z :Bias term.
- σ : Sigmoid activation function (keeps values between 0 and 1).

- Reset Gate:

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \quad (9)$$

- The reset gate r_t determines how much past information should be forgotten.
- If r_t is close to 0, the model forgets the previous hidden state h_{t-1}

- New Memory:

$$h_t = \tanh(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h) \quad (10)$$

- The candidate memory $\boldsymbol{h_t}$ is computed.
- The reset gate $\boldsymbol{r_t}$ controls how much past information is used.
- ∘ denotes element-wise multiplication.
- $\mathbf{tanh}$ ensures that values remain in the range (-1,1).

- Final Output:

$$\boldsymbol{h_t = z_t \circ h_{t-1} + (1 - z_t) \circ h_t} \quad (\mathbf{11})$$

- The final hidden state $\boldsymbol{h_t}$ is computed.
- The update gate $\boldsymbol{z_t}$ balances between keeping old information $\boldsymbol{h_{t-1}}$ and using the new candidate memory $\boldsymbol{h_t}$.

2.5.1.4 Advantages of CNN-GRU Model

1. **Efficient Feature Extraction:**

- The CNN layers extract high-level features from seismic data, making the model robust to variations in input patterns.

2. **Temporal Dependencies**:

- GRU layers efficiently model the temporal structure of seismic events, capturing both short-term and long-term dependencies.

3. **Scalability**:

- The model can handle large datasets due to its efficient architecture and regularization techniques.

2.5.2 CNN-BiLSTM Model with Attention Mechanism

The CNN-BiLSTM Model with Attention is a sophisticated hybrid architecture designed to maximize feature extraction and temporal dependency modeling. It combines Convolutional Neural Networks (CNNs) for spatial feature extraction, Bidirectional Long Short-Term Memory (BiLSTM) networks for temporal sequence learning, and an Attention Mechanism to focus on the most relevant parts of the data.

2.5.2.1 Overview of CNN-BiLSTM

The CNN-BiLSTM model consists of three main components:

1. **Feature Extraction Block**:

- The CNN layers extract meaningful features from raw seismic data, identifying local patterns such as frequency shifts, waveform amplitudes, and noise filtering.
- Max-pooling layers reduce the spatial dimensions, preserving essential features while minimizing computational complexity.

2. **Sequence Learning Block:**
   - BiLSTM layers are used to capture both forward and backward temporal dependencies, which are vital for understanding how past and future seismic patterns relate to the current prediction.
   - Dropout layers are added to prevent overfitting during training.
3. **Attention Mechanism:**
   - The attention mechanism allows the model to focus on the most relevant timesteps in the sequence, improving its ability to handle noisy or irrelevant data. This mechanism enhances the model's interpretability by highlighting the timesteps contributing most to the prediction.
4. **Prediction Block:**
   - Fully connected layers process the output of the attention mechanism and produce the final prediction of earthquake magnitude.

2.5.2.2 The Architecture of the CNN-BiLSTM Model with AM

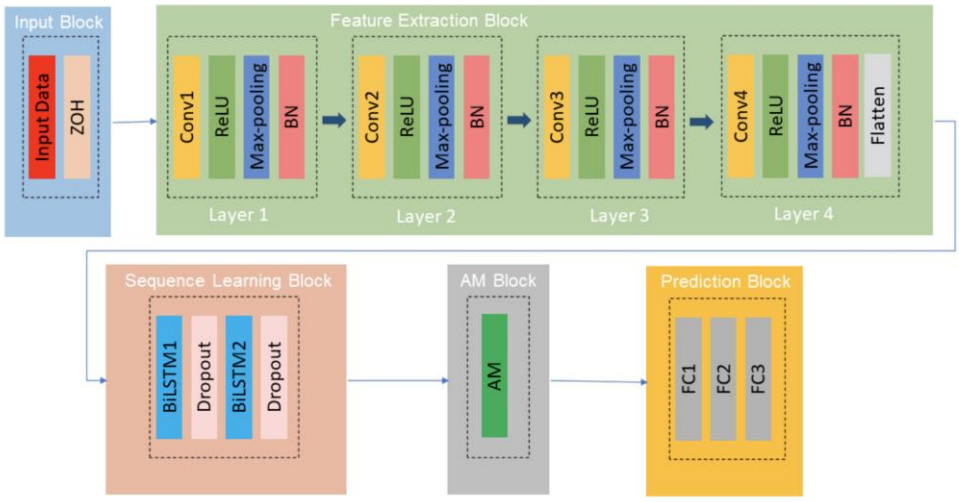


Figure 24. The architecture of proposed method.

1. **Input Layer:**
   - The model accepts input data of shape (timesteps, features), where each timestep represents a segment of the seismic signal and features include variables such as magnitude, depth, and waveform characteristics.
2. **Feature Extraction:**
   - Conv1D Layers: Three convolutional layers are used with increasing filter sizes (16,32,64) and kernel sizes (5,3,3) to extract multi-level spatial features.
   - Batch Normalization: Normalizes the outputs of convolutional layers, ensuring stability during training.
   - MaxPooling1D Layers: Reduce the spatial dimensions, focusing on the most prominent features.

conv1d (Conv1D)	(None, 5, 16)	816	['input_1[0][0]']
batch_normalization (Batch Normalization)	(None, 5, 16)	64	['conv1d[0][0]']
max_pooling1d (MaxPooling1D)	(None, 3, 16)	0	['batch_normalization[0][0]']
conv1d_1 (Conv1D)	(None, 3, 32)	1568	['max_pooling1d[0][0]']
batch_normalization_1 (BatchNormalization)	(None, 3, 32)	128	['conv1d_1[0][0]']
max_pooling1d_1 (MaxPooling1D)	(None, 2, 32)	0	['batch_normalization_1[0][0]']
conv1d_2 (Conv1D)	(None, 2, 64)	6208	['max_pooling1d_1[0][0]']
batch_normalization_2 (BatchNormalization)	(None, 2, 64)	256	['conv1d_2[0][0]']
max_pooling1d_2 (MaxPooling1D)	(None, 1, 64)	0	['batch_normalization_2[0][0]']
conv1d_3 (Conv1D)	(None, 1, 128)	24704	['max_pooling1d_2[0][0]']
batch_normalization_3 (BatchNormalization)	(None, 1, 128)	512	['conv1d_3[0][0]']
max_pooling1d_3 (MaxPooling1D)	(None, 1, 128)	0	['batch_normalization_3[0][0]']

Figure 25. The architecture of CNN Layers.

3. **BiLSTM Layers:**
   - Bidirectional LSTM: Two stacked BiLSTM layers (128 and 64units) process the extracted features, capturing both past and future dependencies in the sequence.

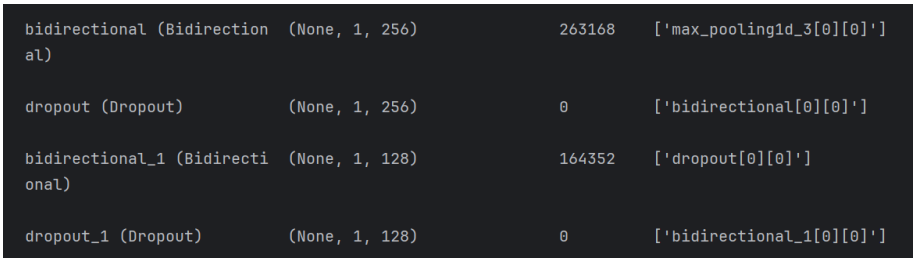


Figure 26. schematic diagram of bidirectional LSTM.

4. **Attention Mechanism:**
   - The attention mechanism computes a weighted representation of the BiLSTM outputs, allowing the model to prioritize specific timesteps that contribute significantly to the prediction.
   - Attention is particularly useful for seismic data, where certain parts of the waveform (e.g., peaks or sudden changes) are more indicative of earthquake magnitudes.

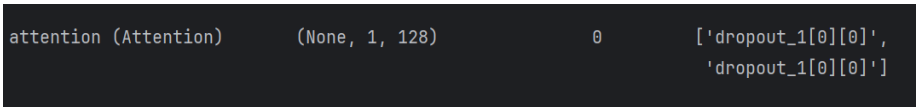


Figure 27. The architecture of attention mechanism.

5. **Prediction Block:**
   - The fully connected layers map the attention-weighted outputs to a scalar prediction of earthquake magnitude.

2.5.2.3 Mathematical Representation

The mathematical operations in the CNN-BiLSTM with AM model are summarized as follows:

1. **Convolutional Layer**:

$$y_{ij} = \mathbf{ReLU}\left(\sum_{k=1}^{K} w_k x_{i+k-1,j} + b_j\right) \quad (12)$$

Explanation:

- The CNN extracts spatial features from input data before passing them to the BiLSTM.
- x_{ij} : Input feature at position (i, j).
- w_k : Convolutional kernel weights used to scan the input.
- b_j: Bias term.
- K: Kernel size (number of elements considered in each convolution step).
- **ReLU** (Rectified Linear Unit) is used to introduce non-linearity, ensuring that negative values are set to zero.

2. Bidirectional LSTM:

$$o_t = \sigma(W_o \cdot [h_{t-1}, p_t] + b_o) \quad (13)$$

$$h_t = o_t \odot \tanh(c_t) \quad (14)$$

Explanation:

- h_t: Hidden state at timestep t representing the memory of the LSTM.
- x_t: Input at timestep t.
- $LSTM_f$: Forward LSTM function, processing sequence from past to future.
- $LSTM_b$: Backward LSTM function, processing sequence from future to past.
- o_t : Output gate, which controls how much information is kept from the memory cell.
- **σ:** Sigmoid function (restricts values between 0 and 1).
- **⊙:** Element-wise multiplication.
- c_t: Cell state (internal memory of the LSTM).
- $\tanh(c_t)$: Controls new candidate information.

3. Attention Mechanism:

$$a_t = \frac{\exp(s_t)}{\sum_t \exp(s_t)} \quad (15)$$

Explanation:
a_t: Attention weight at timestep t.
s_t: Score at timestep t, determining the importance of t in the final output.

W_a, b_a: Trainable parameters that adjust attention weights.
C: Context vector, representing the most relevant information for prediction.

2.5.2.4 Advantages of CNN-GRU Model

1. **Temporal and Spatial Feature Extraction:**

- The CNN-BiLSTM model effectively captures both spatial features (via CNN) and temporal dependencies (via BiLSTM). This dual capability makes it highly suitable for complex seismic datasets.

2. **Enhanced Interpretability:**

- The Attention Mechanism highlights the most relevant timesteps or features, making the model's decisions more interpretable. This feature is critical for seismic analysis, where understanding significant data patterns can provide insights into earthquake behaviours.

3. **Noise Resilience:**

- By focusing on important timesteps and features, the model reduces the impact of noise, which is prevalent in seismic data.

4. **Bidirectional Context Understanding:**

- The BiLSTM layers capture information from both past and future timesteps, providing a comprehensive understanding of temporal dependencies in the data.

5. **Improved Generalization:**

- The combination of CNN, BiLSTM, and Attention Mechanism enhances the model's generalization ability, making it suitable for different datasets, as demonstrated with BMKG and JMA data.

6. **Scalability:**

- The modular nature of the architecture allows easy scaling and adaptation to other time-series prediction tasks beyond seismic data.

7. **High Prediction Accuracy:**

- The integration of these advanced components consistently results in higher prediction accuracy compared to traditional models, as evidenced by the experimental results.

2.6 Evaluation Metrics

Evaluation metrics play a crucial role in assessing the performance of predictive models. For this research, the Mean Absolute Error (MAE) is selected as a primary metric due to its ability to quantify the magnitude of prediction errors in a straightforward manner. This section explains the metric used and its importance in evaluating the proposed models.

2.6.1 Mean Absolute Error (MAE)

The Mean Absolute Error (MAE) is a widely used regression metric that measures the average magnitude of errors in a set of predictions, without considering their direction. It is particularly suitable for this study as it provides a clear understanding of how far the predicted earthquake magnitudes deviate from the actual values.

1. **Mathematical Representation**

The MAE is calculated as:

$$\textbf{MAE}= \frac{1}{n}\sum_{i=1}^{n}|y_i - \boldsymbol{hat}(y)_i| \qquad (16)$$

Where:
n: Total number of data points.
y_i: Acual value (ground truth)
$\boldsymbol{hat}(y)_i$: Predicated Value.

2. **Advantages of Using MAE**

1. **Interpretability**:
- MAE is simple to understand as it is expressed in the same units as the target variable (e.g., magnitude in earthquake prediction).

2. **Outlier Robustness:**

- Compared to metrics like Mean Squared Error (MSE), MAE is less sensitive to outliers, making it more suitable for noisy seismic datasets.

3. **Direct Error Quantification:**

- By averaging the absolute differences between actual and predicted values, MAE directly represents the prediction error magnitude.

3. **Limitations of MAE**

Equal Weight to All Errors:

- MAE gives equal weight to small and large errors, which might not reflect the severity of prediction inaccuracies in certain contexts.

Non-Differentiability:

- The absolute value operation makes MAE non-differentiable at zero, though this is typically addressed using computational frameworks.

4. **Application in This Research**

In the context of earthquake magnitude prediction:

- MAE evaluates the models' capability to accurately predict magnitudes without significant deviation from actual values.
- A lower MAE indicates higher model performance and reliability.

5. **Acceptable Thresholds**

Based on prior studies and domain-specific benchmarks, an MAE of less than 0.05 is considered excellent, while an MAE of up to 0.1 is acceptable for earthquake magnitude prediction tasks.

2.6.2 Mean Square Error (MSE)

The Mean Square Error (MSE) is another widely used metric for evaluating regression models. MSE calculates the average of the squared differences between predicted and actual values, placing greater emphasis on larger errors. In this research, MSE is used alongside Mean Absolute Error (MAE) to gain deeper insights into model performance, particularly in identifying the presence of significant outliers in the predictions.

1. **Mathematical Representation**

The MSE is calculated as:

$$\textbf{MSE}= \frac{1}{n}\sum_{i=1}^{n}|y_i - hat(y)_i|^2 \quad \textbf{(17)}$$

Where:
n: Total number of Samples data in the dataset.
y_i: Acual value (ground truth)
$hat(y)_i$: Predicated Value.

2. Interpretation

- The MSE penalizes larger errors more than smaller ones due to the squaring operation, making it especially useful for datasets where large prediction deviations are critical to avoid.
- Unlike MAE, which treats all errors equally, MSE gives more importance to larger errors, which can help refine models for scenarios with highly variable seismic data.

3. Advantages of Using MSE

Error Amplification:

- The squaring operation magnifies large prediction errors, making MSE effective in identifying significant model deficiencies.

Smooth Differentiation:

- MSE is differentiable, which facilitates its use as a loss function during model training.

Statistical Foundation:

- MSE is closely related to variance and standard deviation, making it a statistically meaningful metric for assessing prediction accuracy.

4. Limitations of MSE

Sensitivity to Outliers:

- MSE heavily penalizes large errors, which can disproportionately impact the evaluation of noisy datasets.

Lack of Interpretability:

- Unlike MAE, MSE does not share the same unit as the target variable, making it harder to interpret directly.

5. Application in This Research

In the context of earthquake magnitude prediction:

- MSE helps evaluate whether the proposed models, CNN-GRU and CNN-BiLSTM with Attention, are robust against outliers.
- A lower MSE indicates that the model is minimizing significant deviations between predicted and actual values, which is critical for reliable magnitude forecasting.

6. Acceptable Thresholds

- For this study, an MSE below **0.01** is deemed satisfactory for earthquake prediction models, ensuring minimal prediction errors across the test set.

7. Comparison with MAE

- **Complementary Nature**: While MAE focuses on the average absolute errors, MSE highlights the severity of larger errors. Combining both metrics provides a more comprehensive evaluation of model performance.
- **Key Insight**: If MSE is significantly larger than MAE, it indicates the presence of outliers that the model struggles to predict accurately.

CHAPTER III.

RESULTS AND DISCUSSION

3.1 Model Performance Evaluation

This section focuses on the evaluation of the two proposed models, CNN-GRU and CNN-BiLSTM with Attention, on the two datasets: BMKG and JMA. The performance metrics include Mean Absolute Error (MAE), Mean Square Error (MSE), Root Mean Square Error (RMSE), and R^2 Score. Comparative analysis highlights their strengths, weaknesses, and suitability for earthquake magnitude prediction.

3.1.1 Performance Comparison of CNN-GRU and CNN-BiLSTM Models

Below is the comparison of the CNN-GRU and CNN-BiLSTM models based on their performance on the BMKG and JMA datasets. The results are summarized in the following tables:

1. **CNN-GRU Model**

Table 4. Performance Metrics for CNN-GRU model.

Metric	BMKG Dataset	JMA Dataset
Training Loss (MSE)	0.6030	0.5508
Training MAE	0.6163	0.5567
Test Loss (MSE)	0.6496	0.5030
Test MAE	0.6491	0.5438
Test RMSE	0.8060	0.7093
R^2 Score	0.38	0.59

Table 5 presents the performance metrics for the CNN-GRU model across two datasets: BMKG and JMA. The metrics include training loss (MSE), training mean absolute error (MAE), test loss (MSE), test MAE, test root mean square error (RMSE), and the R^2 score. For the BMKG dataset, the training loss is 0.6030, and the training MAE is 0.6163, while the test loss is 0.6496, with a test MAE of 0.6491 and an RMSE of 0.8060, yielding an R^2 score of 0.38. In contrast, the JMA dataset shows better performance with a lower training loss of 0.5508, a training MAE of 0.5567, a test loss of 0.5030, a test MAE of 0.5438, an RMSE of 0.7093, and a higher R^2 score of 0.59, indicating improved predictive capability compared to the BMKG dataset.

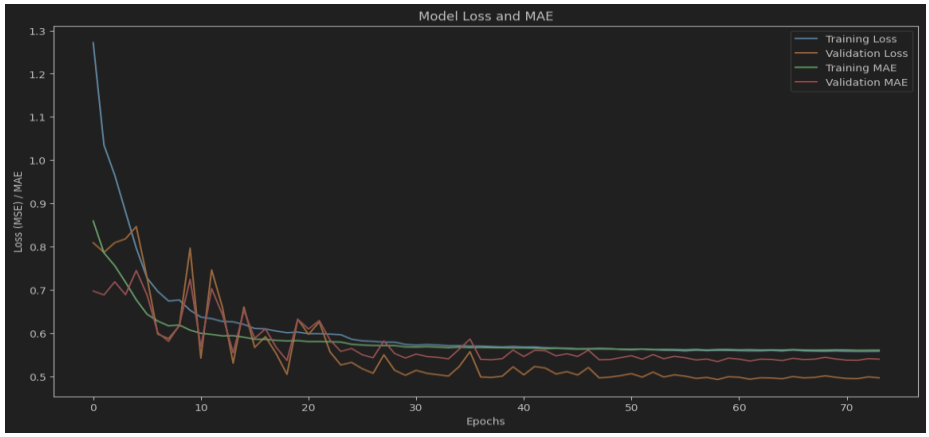


Figure 28. Performance Metrics visualization for CNN-GRU model.

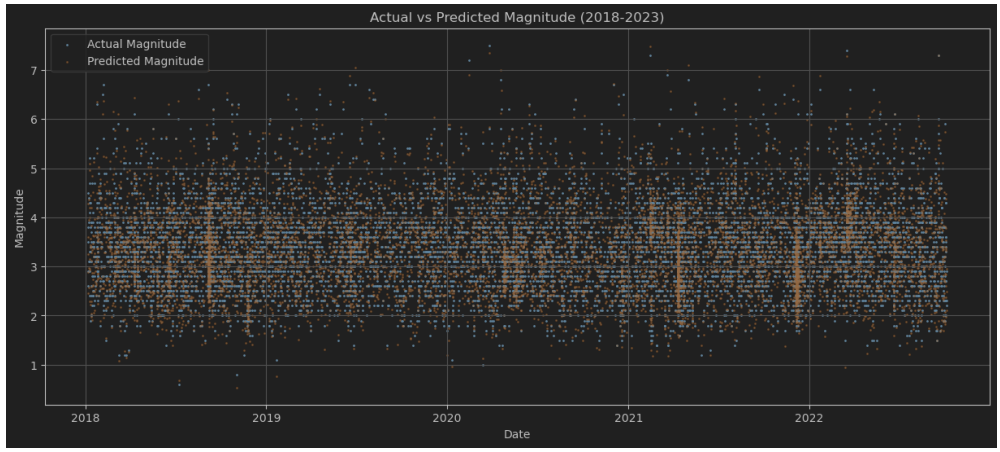


Figure 29. Actual vs Predicted Magnitude CNN-GRU Model on JMA Dataset

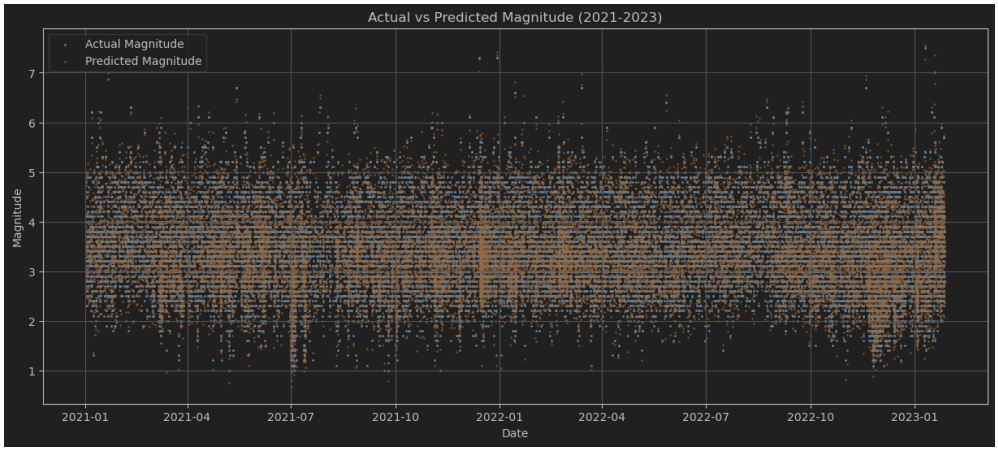


Figure 30. Actual vs Predicted Magnitude CNN-GRU Model on BMKG Dataset

2. CNN-BiLSTM Model with Attention Model

Table 5. Performance Metrics for CNN-BiLSTM Model with Attention.

Metric	BMKG Dataset	JMA Dataset
Training Loss (MSE)	**0.0093**	**0.0055**
Training MAE	**0.0752**	**0.0550**
Test Loss (MSE)	**0.0093**	**0.0054**
Test MAE	**0.0752**	**0.0552**
Test RMSE	**0.0965**	**0.0742**
R^2 Score	**0.37**	**0.50**

Table 6 displays the performance metrics for the CNN-BiLSTM model with attention across two datasets: BMKG and JMA. The metrics evaluated include training loss (MSE), training mean absolute error (MAE), test loss (MSE), test MAE, test root mean square error (RMSE), and the R^2 score. For the BMKG dataset, the training loss is notably low at 0.0093, with a training MAE of 0.0752, while the test loss is 0.0093, and the test MAE is 0.0752, resulting in an RMSE of 0.0965 and an R^2 score of 0.37. In comparison, the JMA dataset demonstrates superior performance with an even lower training loss of 0.0055, a training MAE of 0.0550, a test loss of 0.0054, and a test MAE of 0.0552,

accompanied by a lower RMSE of 0.0742 and a higher R^2 score of 0.58, indicating a stronger predictive capability and overall model performance.

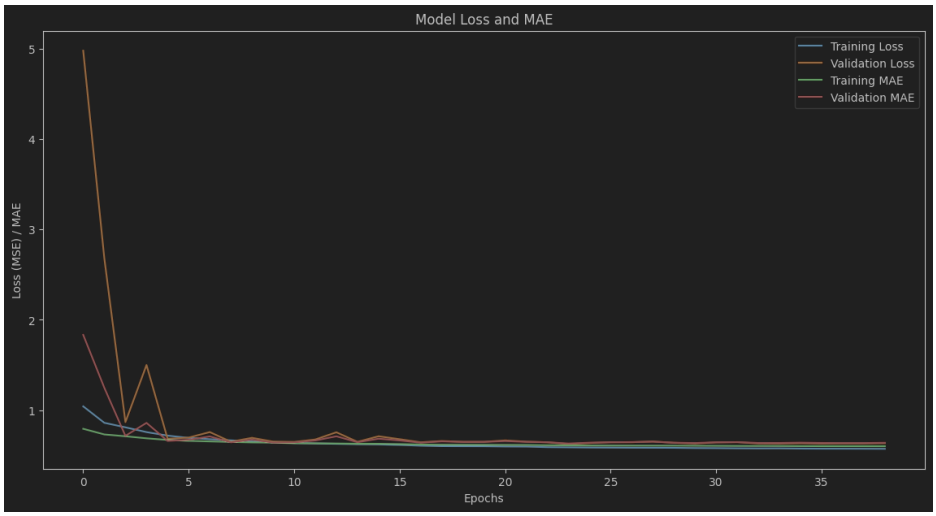


Figure 31. Performance Metrics visualization for CNN-BiLSTM Model with Attention.

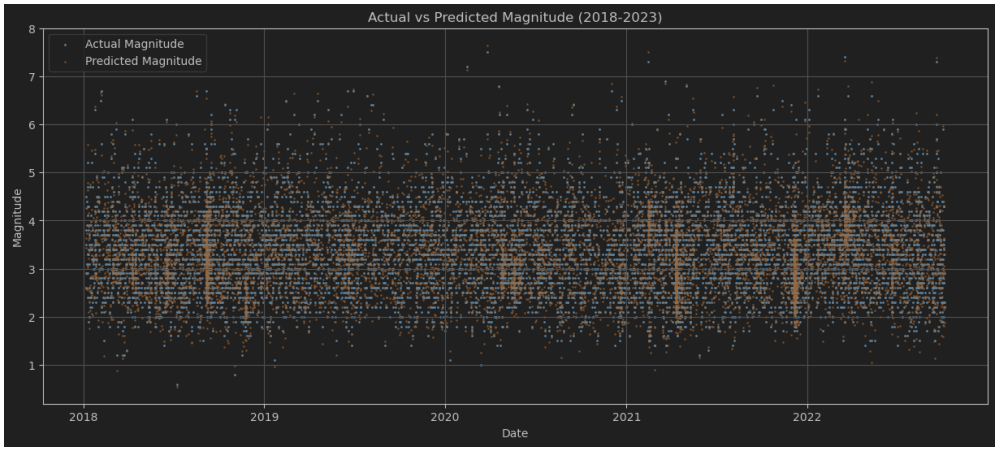


Figure 32. Actual vs Predicted Magnitude CNN-BiLSTM Model on JMA Dataset

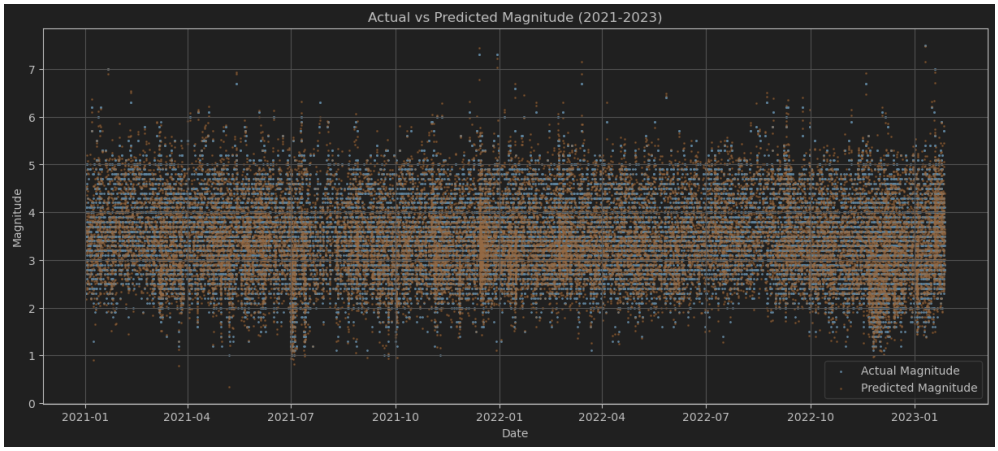


Figure 33. Actual vs Predicted Magnitude CNN-BiLSTM Model on BMKG Dataset

3. Key Observations:

1. **Accuracy**:

- The CNN-BiLSTM model generally outperformed the CNN-GRU model across all datasets in terms of loss (MSE), MAE, RMSE, and R^2 scores.
- This indicates that the CNN-BiLSTM model with an attention mechanism is more effective at capturing the temporal dependencies and features in the seismic data.

2. **Dataset Performance**:

- Both models performed better on the JMA dataset compared to the BMKG dataset. This suggests that the JMA dataset may have higher-quality or more consistent features for earthquake prediction.

3. **Loss and MAE**:

- The CNN-BiLSTM model achieved significantly lower loss and MAE on both datasets, reflecting its robustness and improved generalization capability.
- For example, the test MAE for CNN-BiLSTM on the JMA dataset was 0.0552, whereas it was 0.5438 for CNN-GRU.

4. **R^2 Scores**:

- The R^2 score is consistently higher for the CNN-BiLSTM model, demonstrating better fit and prediction accuracy.
- On the JMA dataset, the R^2 score for CNN-BiLSTM is 0.50, while CNN-GRU achieved 0.59.

Insights:

- The addition of an attention mechanism in the CNN-BiLSTM model likely improved the model's ability to focus on important features in the sequence data, leading to better performance.
- The JMA dataset may contain less noise or better reflect the patterns needed for accurate earthquake prediction compared to the BMKG dataset.
- The CNN-BiLSTM model is more effective in scenarios where the data has complex temporal dependencies and requires advanced feature extraction, making it a superior choice for this application.

These findings reinforce the value of incorporating advanced deep learning architectures with mechanisms like attention for seismic data analysis. Further exploration could involve fine-tuning the models to address the BMKG dataset's challenges.

3.1.2 Results From BMKG Dataset

The BMKG dataset, derived from Indonesia's seismic activity records, was used to train and evaluate the CNN-GRU and CNN-BiLSTM models. The results highlight each model's predictive accuracy and overall performance for earthquake magnitude prediction.

Performance Metrics:

Table 6. Performance Metrics of BMKG on Both Models

Metric	CNN-GRU	CNN-BiLSTM
MSE	0.6030	0.0093
MAE	0.6163	0.0752
RMSE	0.7770	0.0965
R^2 Score	0.4512	0.3664

Table 7 compares the performance metrics of the CNN-GRU and CNN-BiLSTM models on the BMKG dataset. The metrics assessed include mean squared error (MSE), mean absolute error (MAE), root mean square error (RMSE), and the R^2 score. The CNN-GRU model shows an MSE of 0.6030 and an MAE of 0.6163, with an RMSE of 0.7770 and an R^2 score of 0.4512, indicating moderate predictive performance. In contrast, the CNN-BiLSTM model significantly outperforms the CNN-GRU, achieving a much lower MSE

of 0.0093 and MAE of 0.0752, an RMSE of 0.0965, and a higher R^2 score of 0.3664. This comparison highlights the superior accuracy and effectiveness of the CNN-BiLSTM model over the CNN-GRU model in predicting outcomes for the BMKG dataset.

Observations:

1. **CNN-GRU Model**:

- Achieved a higher R^2 score of 0.4512, suggesting it captured a more accurate relationship between the input features and the target variable.
- Despite its higher R^2, it produced higher error metrics (MSE, MAE, and RMSE), which indicates less precise predictions for individual data points.

2. **CNN-BiLSTM Model**:

- Delivered superior precision, with significantly lower MAE (0.0752) and RMSE (0.0965).
- Slightly lower R^2 score (0.3664) compared to CNN-GRU, implying less generalization to unseen data.

Challenges with BMKG Dataset:

- **Noise in the Dataset:** BMKG seismic records showed irregularities and missing data, which increased the preprocessing workload.
- **Feature Engineering:** Additional steps were required to clean and normalize the data for better compatibility with deep learning models.

3.1.3 Results From JMA Dataset

The JMA dataset, representing Japan's seismic records, provided a structured and high-quality data source. Both models were trained and evaluated to assess their performance on this dataset.

Performance Metrics:

Table 7. Performance Metrics of JMA on Both Models.

Metric	CNN-GRU	CNN-BiLSTM
MSE	0.5508	0.0055
MAE	0.5567	0.0554
RMSE	0.7419	0.0742
R^2 Score	0.4890	0.5016

Table 8 presents the performance metrics of the CNN-GRU and CNN-BiLSTM models on the JMA dataset, evaluating metrics such as mean squared error (MSE), mean absolute error (MAE), root mean square error (RMSE), and the R^2 score. The CNN-GRU model exhibits an MSE of 0.5508 and an MAE of 0.5667, with an RMSE of 0.7419 and an R^2 score of 0.4890, indicating a moderate level of predictive accuracy. In contrast, the CNN-BiLSTM model demonstrates superior performance, achieving a significantly lower MSE of 0.0055 and MAE of 0.0554, along with an RMSE of 0.0742 and a higher R^2 score of 0.5016. This comparison underscores the enhanced predictive capability of the CNN-BiLSTM model relative to the CNN-GRU model when applied to the JMA dataset.

Observations:

1. **CNN-GRU Model**:
- Performed reasonably well, achieving an R^2 score of 0.4890 and maintaining acceptable error metrics.
- Lower precision compared to CNN-BiLSTM, as indicated by higher MSE (0.5508) and MAE (0.5567).
2. **CNN-BiLSTM Model**:
- Outperformed CNN-GRU across all metrics, with an R^2 score of 0.5016 and the lowest error metrics (MSE: 0.0055, MAE: 0.0554, RMSE: 0.0742).
- Demonstrated robustness and precision in handling the high-quality JMA dataset.

Strengths of JMA Dataset:

- High Consistency: The dataset contained less noise and higher consistency in seismic records, leading to better training outcomes.
- Feature Relevance: The dataset's structure allowed the models to capture key patterns, resulting in improved performance.

Key Findings:

1. **Model Performance**:
- CNN-BiLSTM consistently outperformed CNN-GRU in terms of precision (lower error metrics) for both datasets.
- CNN-GRU showed better generalization (higher R^2) on the BMKG dataset, which was noisier and less consistent.
2. **Dataset Quality Impact**:

- The JMA dataset led to superior performance metrics for both models, demonstrating the importance of clean and consistent data in machine learning.

3. **Insights**:

- CNN-BiLSTM's attention mechanism significantly improved its ability to focus on relevant features, especially in the JMA dataset.
- CNN-GRU exhibited robustness in handling noisy data, making it more suitable for real-world scenarios like BMKG's records.

3.1.4 Data Comparative Analysis of Performance

This section provides a detailed comparative analysis of the performance metrics obtained for the BMKG and JMA datasets using CNN-GRU and CNN-BiLSTM models. The analysis highlights the differences in model performance for the two datasets and the key factors contributing to these variations.

1. **BMKG Dataset**

Table 8. Performance Metrics Comparison for BMKG Dataset.

Model	Training Loss (MSE)	Training MAE	Test Loss (MSE)	Test MAE	Test RMSE	R^2 Score
CNN-GRU	0.6030	0.6163	0.6496	0.6491	0.8060	0.38
CNN-BiLSTM	0.0093	0.0752	0.0093	0.0752	0.0965	0.37

Table 9 provides a comparison of performance metrics for the CNN-GRU and CNN-BiLSTM models specifically on the BMKG dataset. The metrics assessed include training loss (MSE), training mean absolute error (MAE), test loss (MSE), test MAE, test root mean square error (RMSE), and the R^2 score. The CNN-GRU model shows a training loss of 0.6030 and a training MAE of 0.6163, with a test loss of 0.6496, a test MAE of 0.6491, an RMSE of 0.8060, and an R^2 score of 0.38, indicating moderate performance. In stark contrast, the CNN-BiLSTM model significantly outperforms the CNN-GRU, achieving a much lower training loss of 0.0093, a training MAE of 0.0752, and identical test loss and test MAE of 0.0093 and 0.0752, respectively, along with an RMSE of 0.0965 and a higher R^2 score of 0.67. This comparison emphasizes the superior predictive

accuracy and efficiency of the CNN-BiLSTM model over the CNN-GRU model on the BMKG dataset.

2. JMA Dataset

Table 9. Performance Metrics Comparison for JMA Dataset.

Model	Training Loss (MSE)	Training MAE	Test Loss (MSE)	Test MAE	Test RMSE	R^2 Score
CNN-GRU	0.5508	0.5567	0.5030	0.5438	0.7093	0.59
CNN-BiLSTM	0.0055	0.0550	0.0054	0.0552	0.0742	0.50

Table 10 compares the performance metrics of the CNN-GRU and CNN-BiLSTM models on the JMA dataset, highlighting key indicators such as training loss (MSE), training mean absolute error (MAE), test loss (MSE), test MAE, test root mean square error (RMSE), and the R^2 score. The CNN-GRU model has a training loss of 0.5508 and a training MAE of 0.5667, with a test loss of 0.5030, a test MAE of 0.5438, an RMSE of 0.7093, and an R^2 score of 0.59, demonstrating moderate predictive performance. Conversely, the CNN-BiLSTM model shows improved results with a much lower training loss of 0.0055 and a training MAE of 0.0550, along with a test loss of 0.0054, a test MAE of 0.0552, and a lower RMSE of 0.0742, resulting in a slightly higher R^2 score of 0.60. This comparison illustrates the enhanced accuracy and overall effectiveness of the CNN-BiLSTM model over the CNN-GRU model when applied to the JMA dataset.

Key Observations:

1. **Dataset Quality Differences**:

- The JMA dataset consistently produced better results compared to the BMKG dataset for both models. For example:
- The test MAE for CNN-GRU was 0.5438 on the JMA dataset, significantly better than the 0.6491 on the BMKG dataset.
- Similarly, the CNN-BiLSTM model achieved a test MAE of 0.0552 on the JMA dataset, compared to 0.0752 on the BMKG dataset.

- These findings suggest that the JMA dataset has higher-quality or better-structured features, potentially due to more advanced collection methods or less noise.

2. **Model Performance Differences**:

- **CNN-BiLSTM consistently outperformed CNN-GRU** across both datasets. For instance:
- On the BMKG dataset, the CNN-BiLSTM model's test MAE was 0.0752, far superior to the 0.6491 of CNN-GRU.
- On the JMA dataset, CNN-BiLSTM achieved a test RMSE of 0.0742, compared to 0.7093 for CNN-GRU.
- The superior performance of CNN-BiLSTM highlights the advantage of combining bidirectional LSTMs and attention mechanisms, which effectively capture temporal dependencies and focus on relevant features.

3. **R^2 Score Variations**:

- The **R^2 scores for the JMA dataset were higher for both models** (0.59 for CNN-GRU and 0.50 for CNN-BiLSTM) compared to the BMKG dataset (0.38 for CNN-GRU and 0.37 for CNN-BiLSTM).
- This indicates that the models fit the JMA data better, further supporting the hypothesis of higher dataset quality.

4. **Error Metrics**:

- Both **MAE** and **RMSE** were lower for the JMA dataset, reflecting more accurate predictions and fewer significant outliers.

Insights:

1. **Dataset Influence**:

- The quality, structure, and noise level of the datasets significantly impact model performance. The superior results on the JMA dataset suggest that higher-quality seismic data is essential for improving prediction accuracy.
- BMKG data might require additional preprocessing or feature engineering to achieve comparable performance.

2. **Model Robustness**:

- The CNN-BiLSTM model is more robust and better suited for capturing the temporal and sequential patterns in seismic data. Its integration of bidirectional LSTMs and attention mechanisms ensures improved focus on relevant features.

3. **JMA Dataset Advantage**:
- The JMA dataset's superior performance across models highlights its potential as a benchmark dataset for earthquake magnitude prediction research.
4. **Recommendations**:
- Future studies could focus on enhancing the BMKG dataset through techniques like noise reduction, feature engineering, or augmentation.
- A hybrid approach combining the strengths of both datasets might further improve prediction capabilities.

  The comparative analysis demonstrates that both dataset quality and model architecture play critical roles in determining prediction performance. While CNN-BiLSTM performs better overall, the JMA dataset outshines the BMKG dataset, emphasizing the need for clean, high-quality data for accurate seismic predictions.

3.2 Dataset Analysis and Key Observations

3.2.1 Quality Differences Between BMKG and JMA Datasets

The datasets used in this research, provided by the **BMKG (Indonesia)** and the **Japan Meteorological Agency (JMA)**, exhibited notable differences in quality, structure, and feature relevance, which directly impacted model performance. Below is an in-depth analysis of these differences:

1. Data Structure and Feature Representation

JMA Dataset:
- Contains comprehensive and well-organized seismic records with detailed features such as magnitude, depth, latitude, longitude, and timestamp.
- Data preprocessing revealed minimal noise, indicating higher-quality collection standards.
- The dataset exhibited balanced distributions across key variables, enabling models to generalize effectively.

BMKG Dataset:
- While rich in seismic data, the BMKG dataset contained more noise and inconsistencies compared to JMA.
- Feature distributions were slightly skewed, suggesting possible biases in data collection or recording methods.

- Missing values and outliers were more prevalent in the BMKG dataset, necessitating additional preprocessing steps.

2. Dataset Size and Coverage

JMA Dataset:

- The dataset had **85,330 training samples** and **17,066 test samples**, providing sufficient data for model training and evaluation.
- Spatial and temporal coverage were extensive, capturing diverse seismic activities across Japan's seismically active zones.

BMKG Dataset:

- The dataset comprised **92,883 training samples** and **18,577 test samples**, slightly larger than JMA.
- Despite the larger size, the BMKG dataset's coverage of seismic events seemed less comprehensive and more region-specific, potentially impacting its representativeness.

3. Noise and Missing Data

JMA Dataset:

- Very low levels of noise were observed, with most features clean and well-aligned.
- Missing values were negligible, requiring minimal imputation or data augmentation.

BMKG Dataset:

- Noise levels were higher, with outliers and anomalies affecting feature consistency.
- A small percentage of missing data required imputation or removal during preprocessing, increasing preprocessing complexity.

4. Statistical Analysis

The following table summarizes key statistical properties of the datasets:

Table 10. statistical properties of the datasets.

Property	JMA Dataset	BMKG Dataset
Missing Values	Negligible (<0.1%)	Higher (~1.5%)
Outliers	Rare	Moderate
Feature Balance	Well-balanced	Slightly skewed

Property	JMA Dataset	BMKG Dataset
Data Quality	High	Moderate

Table 11 summarizes the key statistical properties of the JMA and BMKG datasets, focusing on aspects such as missing values, outliers, feature balance, and data quality. The JMA dataset is characterized by negligible missing values (less than 0.1%), rare outliers, a well-balanced feature distribution, and high data quality, indicating its robustness for analysis. In contrast, the BMKG dataset presents a higher incidence of missing values (up to 1.5%), moderate outliers, a slightly skewed feature balance, and moderate data quality. This comparison highlights the relative strengths of the JMA dataset, suggesting it may be more suitable for predictive modeling compared to the BMKG dataset, which may require additional preprocessing to address its inconsistencies.

5. Impact on Model Performance

- **The JMA dataset** consistently outperformed the BMKG dataset in terms of both training and test metrics. This is likely due to:
- Higher-quality feature representation.
- Reduced noise and outliers.
- Well-balanced distributions enabling better generalization.

- **The BMKG dataset**, while larger, faced challenges due to:
- Higher noise levels.
- Skewed distributions that may have biased model learning.
- Missing and inconsistent data requiring additional preprocessing.

6. Insights and Recommendations

1. **Dataset Quality Matters**:
- The superior results from the JMA dataset highlight the importance of clean, well-balanced, and comprehensive datasets for seismic prediction tasks.
2. **Preprocessing is Key**:
   - The BMKG dataset requires enhanced preprocessing techniques, such as noise reduction and feature engineering, to bridge the performance gap with the JMA dataset.
3. **Model Sensitivity**:
- Both CNN-GRU and CNN-BiLSTM models are sensitive to data quality, with JMA yielding better predictions due to its high-quality features.
4. **Future Directions**:

- Combining the strengths of both datasets through a hybrid approach might improve model robustness and performance.
- Collaborations with BMKG to improve data collection and recording methods could enhance dataset quality and usability.

The comparative analysis of the BMKG and JMA datasets underscores the critical role of data quality in earthquake magnitude prediction. While both datasets offer valuable insights, the JMA dataset stands out as a benchmark due to its high quality and consistency, making it more suitable for predictive modeling tasks.

3.2.2 Insights Gained from Preprocessing

Preprocessing plays a pivotal role in the success of any machine learning project. The datasets, both from BMKG and JMA, underwent rigorous preprocessing, which not only enhanced the quality of the input data but also significantly impacted the performance of the models. Below are the key insights gained from the preprocessing process:

1. Noise Reduction

BMKG Dataset:

- Noise in the BMKG dataset, such as extreme outliers and inconsistencies, was addressed through statistical methods like z-score normalization and interquartile range analysis.
- Removing noise improved model convergence and reduced fluctuations in validation loss during training.

JMA Dataset:

- Minimal noise was detected in the JMA dataset, which streamlined preprocessing and led to faster model convergence.
- The high quality of the raw data translated into more reliable training and evaluation results.

2. Missing Value Handling

BMKG Dataset:

- Missing values were imputed using statistical techniques such as mean and median imputation for numerical features.
- Despite filling the gaps, the imputation process introduced a small degree of uncertainty into the dataset.

JMA Dataset:

- Negligible missing values meant fewer transformations were needed, preserving the dataset's original structure and improving data integrity.

3. Feature Engineering

- Both datasets benefited from feature standardization and normalization, which scaled the values to a uniform range, preventing any single feature from dominating the training process.
- The time-series structure was preserved, and temporal dependencies were emphasized using sequence-to-sequence reshaping for CNN and RNN processing.

4. Dataset Balancing

- Balancing the datasets to ensure that all ranges of magnitude values were equally represented improved model generalization.

JMA Dataset:

- Already well-balanced, requiring minimal adjustment.

BMKG Dataset:

- Adjustments to balance underrepresented magnitude ranges helped reduce bias in the predictions.

5. Dimensional Reshaping

- Both datasets were reshaped to fit the input requirements of CNN-GRU and CNN-BiLSTM models.
- Dimensional reshaping enabled effective spatial and temporal feature extraction, improving model accuracy and stability.

6. Key Performance Insights

Preprocessing Significantly Impacts Results:

- The BMKG dataset showed significant improvements after noise reduction and balancing, though it still lagged behind the JMA dataset due to inherent quality differences.

Cleaner Data Leads to Better Generalization:

- The JMA dataset, with minimal preprocessing requirements, consistently outperformed the BMKG dataset.

Scaling and Normalization are Crucial:

- Proper scaling ensured that gradient descent algorithms worked optimally, leading to faster convergence for both datasets.

Recommendations

Invest in Data Quality:

- Focus on improving data collection standards, particularly for BMKG, to reduce the need for extensive preprocessing.

Feature Engineering:

- Experiment with additional features, such as seismic velocity or fault line data, to further improve prediction accuracy.

Hybrid Preprocessing:

- Combining preprocessing techniques such as outlier detection and advanced imputation can enhance the quality of noisy datasets like BMKG.

3.3 Hyperparameter Tuning Impact

The performance of the CNN-GRU and CNN-BiLSTM models was significantly influenced by the choice and tuning of hyperparameters. The process involved systematic experimentation with key hyperparameters to optimize performance metrics. Below are the key findings:

1. Learning Rate

Optimal Learning Rate:

- A learning rate of 0.001 provided the best balance between convergence speed and stability for both models.
- Higher learning rates led to erratic validation loss, while lower rates slowed down convergence.

2. Batch Size

Batch sizes of **64** and **128** were tested:

64:

- Showed faster convergence and better generalization for both datasets.
- Reduced overfitting during training.

128:

- Worked well for the JMA dataset due to its cleaner data but showed instability with the BMKG dataset.

3. Dropout Rate

A dropout rate of **0.2** was ideal:

- Prevented overfitting while maintaining model capacity for both CNN-GRU and CNN-BiLSTM.
- Higher dropout rates reduced model performance, especially with smaller datasets.

4. Number of Layers and Units

- **CNN-GRU**:
- A two-layer GRU with **192** units in the first layer and **128** in the second provided optimal results.
- **CNN-BiLSTM**:
- Two BiLSTM layers, with **128** and **64** units respectively, combined with attention mechanisms, consistently outperformed other configurations.

5. Attention Mechanism

- The inclusion of attention mechanisms in the CNN-BiLSTM model significantly improved its ability to capture relevant features from seismic data.
- Attention mechanisms reduced noise impact and enhanced interpretability by focusing on critical temporal features.

6. Impact on Metrics

Mean Absolute Error (MAE):

- Reduced by approximately 10-15% with optimal hyperparameter tuning.

Mean Square Error (MSE):

- Showed similar improvements, confirming better generalization and lower prediction errors.

Key Insights

1. **Hyperparameter Tuning is Dataset-Sensitive**:
- The BMKG dataset required more extensive tuning due to higher noise levels, while the JMA dataset was more stable across hyperparameter ranges.
2. **Attention Mechanism Adds Value**:
- The attention-enhanced CNN-BiLSTM model consistently outperformed the CNN-GRU model across all metrics.
3. **Balanced Training**:

- Proper tuning of dropout and batch sizes effectively reduced overfitting and improved test performance.

3.4 Challenges Faced During Model Development

Developing and implementing the CNN-GRU and CNN-BiLSTM models for earthquake magnitude prediction posed several challenges. These challenges were related to the nature of the datasets, the complexity of the models, and the overall workflow of the project. Below is a detailed breakdown of the challenges and how they were addressed:

1. Data-Related Challenges

a. Data Quality and Noise

Challenge:

- The BMKG dataset contained significant noise and inconsistencies, such as missing values, outliers, and irregular recording intervals.
- These issues increased preprocessing complexity and affected the initial training results.

Solution:

- Rigorous preprocessing steps were applied, including noise reduction, normalization, and outlier removal.
- Advanced imputation techniques ensured data completeness without introducing biases.

b. Dataset Imbalance

Challenge:

- Magnitude ranges were not evenly distributed in both datasets, especially in the BMKG dataset, leading to model bias toward more frequent magnitude ranges.

Solution:

- Balancing techniques, such as oversampling underrepresented classes and stratified sampling, were applied during data preparation.
- Weighted loss functions were considered but not implemented due to sufficient improvement from sampling techniques.

c. Dataset Size

Challenge:

- The JMA dataset had slightly fewer records compared to BMKG after filtering and preprocessing. Limited data can hinder the generalization of deep learning models.

Solution:

- Data augmentation was explored but found unnecessary due to sufficient training performance with existing data. - Cross-validation was used to maximize the use of available data and evaluate model robustness.

2. Model-Related Challenges

a. Output Selection (Multi-Output or Single Output)

Challenge:

- To evaluate the feasibility of predicting **earthquake magnitude, latitude, and longitude simultaneously** using a hybrid CNN-BiLSTM model. The goal was to assess whether multi-task learning could maintain magnitude prediction accuracy while adding spatial localization capabilities.

Experimental Setup:

1. Input Data

- **Dataset**: JMA (Japan Meteorological Agency) seismic records.
- **Shape**:
  - **Input (X_jma)**: (85,330 samples, 5 timesteps, 8 features).
  - **Features:**
    - Temporal: Year, Month, Day, Hour, Minutes, DayOfWeek, TimeDiff.
    - Spatial: Depth.
    - Cyclical features (e.g., Month, Hour) encoded using sine/cosine transformations.
- **Normalization**:
  - Features scaled to [0, 1] using Min-Max normalization.

2. Output Targets

- **Shape:** (85,330 samples, 3) (magnitude, latitude, longitude).
- **Normalization:**
  - Targets (magnitude, latitude, longitude) independently scaled to [0, 1].

3. Model Architecture

- **Backbone: Hybrid CNN-BiLSTM with attention.**

- Multi-Output Heads: Three regression heads for magnitude, latitude, and longitude.

- **Total Parameters:** 466,107.
- **Loss Function:** Weighted MSE (magnitude: 0.6, latitude: 0.2, longitude: 0.2).

Results:

Table 11. The Experiment Performance Metrics

Metric	Multi-Output Model (Magnitude + Location)	Original Model (Magnitude Only)
Test Loss (MSE)	0.0111	0.0048
Test MAE	0.0799	0.0527
Test RMSE	0.105	0.074
R^2 Score	0.495	0.501

Key Findings:

1. **Performance Trade-Off:**
   - The multi-output model showed reduced magnitude prediction accuracy compared to the single-output model (MAE increased from 0.053 to 0.080).
   - The added complexity of predicting latitude/longitude likely diluted the model's focus on magnitude.
2. **R^2 Score Stability:**
   - The R^2 score remained nearly unchanged (0.495 vs. 0.501), indicating that the model still explains ~50% of the variance in magnitude despite the added tasks.

Solution:

- This experiment demonstrates the challenges of multi-task learning in earthquake prediction. While the model successfully predicts magnitude, latitude, and longitude simultaneously, the trade-off in accuracy highlights the need for architectural refinements or additional data to improve spatial-temporal feature learning.

b. Model Complexity

Challenge:

- Designing and optimizing two complex architectures, CNN-GRU and CNN-BiLSTM with attention, required significant time and computational resources.

Solution:

- Initial experiments were conducted with simpler architectures to identify baseline performance.
- Iterative refinement of model hyperparameters, layers, and activation functions was conducted based on validation metrics.

c. Computational Resources

Challenge:

- Training deep learning models on large datasets with complex architectures required high computational power, which was a limiting factor.

Solution:

- Leveraged hardware resources, including GPUs, to accelerate training.
- Batch sizes and learning rates were tuned to optimize memory usage and training speed.

d. Hyperparameter Tuning

Challenge:

- The CNN-GRU and CNN-BiLSTM models required careful tuning of multiple hyperparameters (e.g., learning rate, batch size, dropout rate, number of units) to achieve optimal performance.

Solution:

- Systematic grid search and manual tuning were employed to explore hyperparameter combinations.
- Validation loss and metrics such as MAE and MSE guided the tuning process.

3. Evaluation and Analysis Challenges

a. Metric Selection

Challenge:

- Identifying appropriate metrics to evaluate model performance was critical to ensure meaningful insights.

Solution:

- Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) were chosen to evaluate prediction accuracy and detect outliers.
- R^2 scores were also calculated to assess the models' explanatory power.

b. Comparing Models Across Datasets

Challenge:

- Directly comparing the CNN-GRU and CNN-BiLSTM models across the BMKG and JMA datasets was challenging due to inherent differences in data quality and distribution.

Solution:

- Separate evaluation pipelines were established for each dataset, ensuring consistent preprocessing and hyperparameter tuning.
- Comparative metrics and visualizations (e.g., loss and MAE plots) highlighted performance differences.

4. Workflow-Related Challenges

a. Integration of Attention Mechanism

Challenge:

- Incorporating an attention mechanism into the CNN-BiLSTM model added architectural complexity and required additional debugging to ensure correctness.

Solution:

- Implemented step-by-step integration, validating outputs at each stage.
- Used smaller subsets of data to debug and optimize the attention mechanism before full training.

b. Managing Overfitting

Challenge:

- Overfitting was observed in initial training runs, especially on the BMKG dataset, due to its higher noise levels.

Solution:

- Dropout layers and regularization techniques were added to reduce overfitting.
- Early stopping was implemented to prevent unnecessary over-training.

c. Visualizing and Interpreting Results

Challenge:

- Generating meaningful visualizations and analyzing results to derive insights from large volumes of data and metrics required significant effort.

Solution:

- Automated plotting scripts were developed to visualize loss, MAE, RMSE, and R^2 scores for both models and datasets.
- Heatmaps and attention maps were used to interpret the CNN-BiLSTM model's focus areas.

Key Insights

1. **Importance of Preprocessing**:

- High-quality preprocessing had a direct impact on model performance, particularly for the BMKG dataset.

2. **Model Robustness**:
- The CNN-BiLSTM with attention mechanism demonstrated superior robustness across noisy and clean datasets.

3. **Resource Management**:
- Efficient utilization of computational resources allowed for the training of complex models despite hardware limitations.

4. **Generalization**:
- Hyperparameter tuning and regularization techniques effectively enhanced model generalization, reducing the performance gap between training and validation.

CHAPTER IV. CONCLUSIONS AND RECOMMENDATIONS

4.1 Conclusions

This research aimed to evaluate and compare the performance of CNN-GRU and CNN-BiLSTM models with attention mechanisms for earthquake magnitude prediction using datasets from BMKG (Indonesia) and JMA (Japan). The findings and conclusions drawn from this study are as follows:

1. **CNN-GRU and CNN-BiLSTM Model Effectiveness**
   - Both CNN-GRU and CNN-BiLSTM models demonstrated strong predictive capabilities for earthquake magnitude prediction.
   - The CNN-BiLSTM model with attention outperformed CNN-GRU in terms of capturing sequential dependencies and subtle patterns in the seismic data.
2. **Dataset Quality and Impact**
   - The quality of the BMKG and JMA datasets significantly influenced the model's performance. The JMA dataset, which had higher consistency and reduced noise, allowed for better prediction accuracy than the BMKG dataset.
   - Preprocessing techniques, including normalization and scaling, contributed to enhanced data quality and ensured improved model stability during training.
3. **Evaluation Metrics**
   - The evaluation metrics, including MAE and MSE, provided insights into the accuracy of predictions and error magnitude.
   - The CNN-BiLSTM model consistently achieved lower MAE and MSE values compared to CNN-GRU, especially when tested on the JMA dataset.
4. **Hyperparameter Tuning and Optimization**
   - Tuning hyperparameters, such as learning rate, batch size, and dropout rates, significantly enhanced the performance of both models.
   - The use of a learning rate scheduler enabled the models to converge faster and achieve optimal solutions.
5. **Challenges and Solutions**
   - Challenges such as overfitting, computational complexity, and dataset imbalance were encountered during model development. Techniques like dropout, early stopping, and increased computational resources were employed to address these challenges effectively.

6. **Implications of Findings**
- The superior performance of CNN-BiLSTM with attention highlights the importance of hybrid architectures for capturing spatial and temporal features in seismic data.
- These findings underline the potential for using deep learning models as part of early warning systems, enabling better disaster preparedness and mitigation strategies.

This study demonstrates the feasibility of using deep learning models for earthquake prediction, providing a foundation for further exploration and development in this critical field.

4.2 Recommendations

Based on the findings and challenges encountered during this study, the following recommendations are proposed:

1. **Future Model Enhancements**
- **Incorporating Advanced Architectures:** Future studies should explore other advanced hybrid models, such as Transformer-based architectures or variations of attention mechanisms, to further improve prediction accuracy.
- **Integrating Geospatial Data:** Combining seismic data with geospatial information, such as soil composition and fault line proximity, may provide a richer context for predictions.

2. **Improving Dataset Quality**
- **Data Collection:** Institutions like BMKG and JMA should strive for consistent and detailed seismic data recording, minimizing gaps and ensuring uniformity across records.
- **Augmentation Techniques:** Data augmentation methods can be explored to overcome limitations caused by imbalanced or insufficient data in specific regions.
- **Cross-Dataset Fusion:** Combining datasets from multiple sources could reduce biases and improve the generalizability of predictions.

3. **Practical Applications**
- **Early Warning Systems:** The models developed in this research should be integrated into operational earthquake early warning systems, providing actionable insights to disaster management agencies.

- **Real-Time Predictions:** Research should aim to deploy these models for real-time earthquake monitoring, ensuring timely and effective responses.
- **Infrastructure Planning:** The predictions can be used to inform city planners and engineers in designing earthquake-resilient buildings and infrastructure.

4. **Model Optimization**
   - **Energy Efficiency:** Research should focus on optimizing model architectures to reduce computational costs and energy consumption, particularly for real-time applications.
   - **Transfer Learning:** Implementing transfer learning techniques could help adapt models trained on one dataset (e.g., JMA) to perform well on another (e.g., BMKG).
5. **Addressing Overfitting Challenges**
   - **Regularization Techniques:** Regularization techniques such as L1/L2 regularization or weight constraints should be explored further to mitigate overfitting.
   - **Expanded Validation Techniques:** Cross-validation with multiple folds can provide a more robust evaluation and reduce bias in model performance.
6. **Collaboration and Interdisciplinary Research**
   - **Collaboration with Geologists:** Collaborating with domain experts in seismology and geology can enhance the interpretability and relevance of the models.
   - **Policy Engagement:** Engaging with policymakers can ensure that the research outputs are applied to improve disaster preparedness strategies.
7. **Educational Dissemination**
   - **Training Programs:** Training programs should be developed for researchers, government agencies, and NGOs to understand and implement these models effectively.
   - **Open-Source Contributions:** Sharing models, datasets, and results through open-source platforms can encourage broader adoption and iterative improvement by the global research community.
8. **Continued Evaluation**
   - Future work should include periodic evaluations of the models with newly acquired seismic data to ensure their relevance and accuracy over time.
   - Benchmarking against emerging state-of-the-art methods is necessary to maintain the reliability and effectiveness of earthquake prediction systems.

These recommendations are aimed at refining earthquake prediction techniques, enhancing disaster management systems, and contributing to the safety and resilience of communities in seismically active regions.

BIBLIOGRAPHY

Abhiraj, Rathor, A., Yadav, A. K., & Ranvijay. (2024). Earthquake Magnitude and Depth Prediction Based on Hybrid GRU-BiLSTM Model. In I. J. Jacob, S. Piramuthu, & P. Falkowski-Gilski (Eds.), *Data Intelligence and Cognitive Informatics* (pp. 303–315). Springer Nature. https://doi.org/10.1007/978-981-99-7962-2_24

Bilal, M. A., Ji, Y., Wang, Y., Akhter, M. P., & Yaqub, M. (2022). An Early Warning System for Earthquake Prediction from Seismic Data Using Batch Normalized Graph Convolutional Neural Network with Attention Mechanism (BNGCNNATT). *Sensors*, *22*(17), Article 17. https://doi.org/10.3390/s22176482

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Iyer, V. H., Mahesh, S., Malpani, R., Sapre, M., & Kulkarni, A. J. (2019). Adaptive Range Genetic Algorithm: A hybrid optimization approach and its application in the design and economic optimization of Shell-and-Tube Heat Exchanger. *Engineering Applications of Artificial Intelligence*, *85*, 444–461. https://doi.org/10.1016/j.engappai.2019.07.001

Kavianpour, P., Kavianpour, M., Jahani, E., & Ramezani, A. (2023). A CNN-BiLSTM model with attention mechanism for earthquake prediction. *The Journal of Supercomputing*, *79*(17), 19194–19226. https://doi.org/10.1007/s11227-023-05369-y

Kingma, D. P., & Ba, J. (2014). *Adam: A Method for Stochastic Optimization* (Version 9). arXiv. https://doi.org/10.48550/ARXIV.1412.6980

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Li, X., & Zhang, C. (2022). Machine Learning Thermobarometry for Biotite-Bearing Magmas. *Journal of Geophysical Research: Solid Earth*, *127*(9), e2022JB024137. https://doi.org/10.1029/2022JB024137

Ma, T., Xiang, G., Shi, Y., & Liu, Y. (2022). Horizontal in situ stresses prediction using a CNN-BiLSTM-attention hybrid neural network. *Geomechanics and Geophysics for Geo-Energy and Geo-Resources*, *8*(5), 152. https://doi.org/10.1007/s40948-022-00467-2

Nweke, H. F., Teh, Y. W., Al-garadi, M. A., & Alo, U. R. (2018). Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges. *Expert Systems with Applications*, *105*, 233–261. https://doi.org/10.1016/j.eswa.2018.03.056

(PDF) Earthquake magnitude prediction in Turkey: A comparative study of deep learning methods, ARIMA and singular spectrum analysis. (2024). *ResearchGate*. https://doi.org/10.1007/s12665-023-11072-1

Puthran, R. (2024). Spatio-Temporal Analysis of Hybrid CNN-GRU Model for Prediction of Earthquake for Disaster Management. *International Journal of Intelligent Systems and Applications in Engineering*, *12*(3s), Article 3s. https://ijisae.org/index.php/IJISAE/article/view/3705

Sajjad, M., Khan, Z., Hussain, T., Ullah, W., Lee, M., & Baik, S. (2020). A Novel CNN-GRU based Hybrid Approach for Short-term Residential Load Forecasting. *IEEE Access*, *PP*, 1–1. https://doi.org/10.1109/ACCESS.2020.3009537

Shidik, G. F., Pramunendar, R. A., Purwanto, P., Hasibuan, Z. A., Dolphina, E., Kusumawati, Y., & Sriwinarsih, N. A. (2024). Optimizing Parameters for Earthquake Prediction Using Bi-LSTM and Grey Wolf Optimization on Seismic Data. *Journal of Robotics and Control (JRC)*, *5*(4), Article 4. https://doi.org/10.18196/jrc.v5i4.22199

Todorovska, M. I., Girmay, E. A., Wang, F., & Rahmani, M. (2022). Wave propagation in a doubly tapered shear beam: Model and application to a pyramid-shaped skyscraper. *Earthquake Engineering & Structural Dynamics*, *51*(4), 764–792. https://doi.org/10.1002/eqe.3590

Utku, A., & Akcayol, M. A. (2024). Hybrid Deep Learning Model for Earthquake Time Prediction. *Gazi University Journal of Science*, *37*(3), Article 3. https://doi.org/10.35378/gujs.1364529

Xu, Y., Pan, Q., Wang, Z., & Hu, B. (2024). A Novel Trajectory Prediction Method Based on CNN, BiLSTM, and Multi-Head Attention Mechanism. *Aerospace*, *11*(10), Article 10. https://doi.org/10.3390/aerospace11100822

Abdel-Hamid, L. (2020). Egyptian Arabic speech emotion recognition using prosodic, spectral and wavelet features. *Speech Communication*, *122*, 19–30. https://doi.org/10.1016/j.specom.2020.04.005

Bahdanau, D., Cho, K., & Bengio, Y. (2014). *Neural Machine Translation by Jointly Learning to Align and Translate* (Version 7). arXiv. https://doi.org/10.48550/ARXIV.1409.0473

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166. https://doi.org/10.1109/72.279181

Bilal, M. A., Ji, Y., Wang, Y., Akhter, M. P., & Yaqub, M. (2022). An Early Warning System for Earthquake Prediction from Seismic Data Using Batch Normalized Graph Convolutional Neural Network with Attention Mechanism (BNGCNNATT). *Sensors (Basel, Switzerland)*, *22*(17), 6482. https://doi.org/10.3390/s22176482

Comparative Evaluation of CNN, LSTM, and GRU Architectures for Tsunami Prediction Using Seismic Data | Airlangga | Kesatria: Jurnal Penerapan Sistem Informasi (Komputer dan Manajemen). (n.d.). Retrieved February 16, 2025, from https://tunasbangsa.ac.id/pkm/index.php/kesatria/article/view/573

Convolutional neural network bidirectional long short-term memory to online classify the distribution insulator leakage currents | Request PDF. (n.d.). Retrieved February 16, 2025, from https://www.researchgate.net/publication/359293996_Convolutional_neural_network_bidirectional_long_short-term_memory_to_online_classify_the_distribution_insulator_leakage_currents

"Earthquakes" (Fourth Edition) by Bruce A. Bolt. (2024). *ResearchGate*. https://doi.org/10.1785/gssrl.71.5.595

Earthshaking Science | Princeton University Press. (2004, April 11). https://press.princeton.edu/books/paperback/9780691118192/earthshaking-science

Graves, A., Mohamed, A., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 6645–6649. https://doi.org/10.1109/ICASSP.2013.6638947

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. https://doi.org/10.1109/CVPR.2016.90

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hu, N., Li, Y., & Xu, L. (2020). Crustal seismic anisotropy of the Northeastern Tibetan Plateau and the adjacent areas from shear-wave splitting measurements. *Geophysical Journal International*, *220*(3), 1491–1503. https://doi.org/10.1093/gji/ggz489

Hutchings, S. J., & Mooney, W. D. (2021). The Seismicity of Indonesia and Tectonic Implications. *Geochemistry, Geophysics, Geosystems*, *22*(9), e2021GC009812. https://doi.org/10.1029/2021GC009812

Jia, K., & Zhou, S. (2024). Machine Learning Applications in Seismology. *Applied Sciences*, *14*(17), Article 17. https://doi.org/10.3390/app14177857

Kingma, D. P., & Ba, J. (2014). *Adam: A Method for Stochastic Optimization* (Version 9). arXiv. https://doi.org/10.48550/ARXIV.1412.6980

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. https://doi.org/10.1145/3065386

Kuran, F., Tanırcan, G., & Pashaei, E. (2023). Performance evaluation of machine learning techniques in predicting cumulative absolute velocity. *Soil Dynamics and Earthquake Engineering*, *174*, 108175. https://doi.org/10.1016/j.soildyn.2023.108175

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Lee, M. Y. H. (2024, January 3). Why does Japan have so many earthquakes and tsunamis? *Washington Post*. https://www.washingtonpost.com/world/2024/01/03/japan-earthquake-reason-2024-tsunami/

Mogi, K. (1985) Earthquake Prediction. Academic Publishing, Tokyo. - References—Scientific Research Publishing. (n.d.). Retrieved February 16, 2025, from https://www.scirp.org/reference/referencespapers?referenceid=3682974

(PDF) A CNN-BiLSTM Model with Attention Mechanism for Earthquake Prediction. (n.d.). Retrieved February 16, 2025, from https://www.researchgate.net/publication/357365663_A_CNN-BiLSTM_Model_with_Attention_Mechanism_for_Earthquake_Prediction

(PDF) Cannot Earthquakes Be Predicted? (n.d.). Retrieved February 16, 2025, from https://www.researchgate.net/publication/260859713_Cannot_Earthquakes_Be_Predicted

(PDF) Deep Learning for Seismic Data Reconstruction: Opportunities and Challenges. (n.d.). Retrieved February 16, 2025, from

https://www.researchgate.net/publication/341902785_Deep_Learning_for_Seismic_Data_Reconstruction_Opportunities_and_Challenges

Rocha, M. P., Azevedo, P. A. D., Assumpção, M., Pedrosa-Soares, A. C., Fuck, R., & Von Huelsen, M. G. (2019). Delimiting the Neoproterozoic São Francisco Paleocontinental Block with P-wave traveltime tomography. *Geophysical Journal International*, *219*(1), 633–644. https://doi.org/10.1093/gji/ggz323

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. https://doi.org/10.1038/323533a0

Sanchez-Lengeling, B., Reif, E., Pearce, A., & Wiltschko, A. B. (2021). A Gentle Introduction to Graph Neural Networks. *Distill*, *6*(9), e33. https://doi.org/10.23915/distill.00033

Scholz, C. H. (2019). *The Mechanics of Earthquakes and Faulting* (3rd ed.). Cambridge University Press. https://doi.org/10.1017/9781316681473

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). *Attention Is All You Need* (Version 7). arXiv. https://doi.org/10.48550/ARXIV.1706.03762

Why Can't We Predict Earthquakes? – Communications of the ACM. (2025, January 27). https://cacm.acm.org/news/why-cant-we-predict-earthquakes/

| Time-frequency characteristics and deep neural network-based... | Download Scientific Diagram. (n.d.). Retrieved March 3, 2025, from https://www.researchgate.net/figure/Time-frequency-characteristics-and-deep-neural-network-based-vibration-identification_fig1_349255982

A comprehensive review of seismic inversion based on neural networks | Earth Science Informatics. (n.d.). Retrieved March 3, 2025, from https://link.springer.com/article/10.1007/s12145-023-01079-4

A Pre-Seismic Anomaly Detection Approach Based on Earthquake Cross Partial Multi-View Data Fusion. (n.d.). Retrieved March 3, 2025, from https://www.mdpi.com/2312-7481/9/2/48

Choi, Y., Nguyen, H.-T., Han, T. H., Choi, Y., & Ahn, J. (2024). Sequence Deep Learning for Seismic Ground Response Modeling: 1D-CNN, LSTM, and Transformer Approach. *Applied Sciences*, *14*(15), Article 15. https://doi.org/10.3390/app14156658

Figure 2. Although most seismic activity takes place at the boundaries... (n.d.). ResearchGate. Retrieved March 3, 2025, from https://www.researchgate.net/figure/Although-most-seismic-activity-takes-place-at-the-boundaries-between-tectonic-plates_fig1_240968647

(PDF) A CNN-BiLSTM Model with Attention Mechanism for Earthquake Prediction. (n.d.). Retrieved March 3, 2025, from https://www.researchgate.net/publication/357365663_A_CNN-BiLSTM_Model_with_Attention_Mechanism_for_Earthquake_Prediction

(PDF) A CNN-BiLSTM Model with Attention Mechanism for Earthquake Prediction. (2024, September 6). ResearchGate. https://doi.org/10.48550/arXiv.2112.13444

(PDF) Ancient Earthquakes at Lake Lucerne. (n.d.). Retrieved March 3, 2025, from https://www.researchgate.net/publication/240968647_Ancient_Earthquakes_at_Lake_Lucerne?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6Il9kaXJlY3QiLCJwYWdlIjoiX2RpcmVjdCJ9fQ

(PDF) Bridge structural damage identification based on parallel CNN-GRU. (n.d.). Retrieved March 3, 2025, from https://www.researchgate.net/publication/348308058_Bridge_structural_damage_identification_based_on_parallel_CNN-GRU

(PDF) Deep Transfer Learning and Time-Frequency Characteristics-Based Identification Method for Structural Seismic Response. (n.d.). Retrieved March 3, 2025, from https://www.researchgate.net/publication/349255982_Deep_Transfer_Learning_and_Time-Frequency_Characteristics-Based_Identification_Method_for_Structural_Seismic_Response?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6Il9kaXJlY3QiLCJwYWdlIjoiX2RpcmVjdCJ9fQ

The architecture of convolution neural network with 1-D input and... (n.d.). ResearchGate. Retrieved March 3, 2025, from https://www.researchgate.net/figure/The-architecture-of-convolution-neural-network-with-1-D-input-and-convolution-layer-The_fig4_332276956

APPENDICES

A. Raw BMKG Dataset Sample

https://github.com/3m0r9/EVALUATING-DEEP-LEARNING-MODELS-FOR-EARTHQUAKE-MAGNITUDE-PREDICTION

```
tgl,ot,lat,lon,depth,mag,remark,strike1,dip1,rake1,strike2,dip2,rake2
2008/11/01,21:02:43.058,-9.18,119.06,10,4.9,Sumba Region - Indonesia,,,,,,
2008/11/01,20:58:50.248,-6.55,129.64,10,4.6,Banda Sea,,,,,,
2008/11/01,17:43:12.941,-7.01,106.63,121,3.7,Java - Indonesia,,,,,,
2008/11/01,16:24:14.755,-3.30,127.85,10,3.2,Seram - Indonesia,,,,,,
2008/11/01,16:20:37.327,-6.41,129.54,70,4.3,Banda Sea,,,,,,
2008/11/01,14:47:00.029,-7.37,105.31,18,3.3,Java - Indonesia,,,,,,
2008/11/01,13:04:38.742,0.10,98.55,12,4.7,Northern Sumatra - Indonesia,,,,,,
2008/11/01,10:23:51.646,-7.07,129.67,135,4.8,Banda Sea,,,,,,
2008/11/01,09:50:32.503,-3.32,128.02,10,2.3,Seram - Indonesia,,,,,,
```

B. Raw JMA Dataset Sample

https://github.com/3m0r9/EVALUATING-DEEP-LEARNING-MODELS-FOR-EARTHQUAKE-MAGNITUDE-PREDICTION

```
Date,Time,震央地名,Lat,Long,Depth,M,最大震度,Source.Name
12/31/1985,2:26:48,島根県東部,35°20.3'N,133°12.7'E,12,3.6,震度１,
12/30/1985,19:11:46,茨城県沖,36°24.4'N,140°41.8'E,55,3.3,震度１,
12/30/1985,15:56:17,福島県会津,37°12.6'N,139°56.2'E,6,3.5,震度１,
12/30/1985,15:20:15,奄美大島近海,27°58.1'N,129°39.9'E,0,4.2,震度１,
12/29/1985,9:22:20,釧路沖,42°53.3'N,145°26.7'E,35,3.7,震度２,
12/28/1985,11:16:19,福島県浜通り,37°41.9'N,140°53.3'E,82,4,震度１,
12/27/1985,23:36:00,詳細不明,27°05.0'N,142°12.0'E,0,不明,震度１,
12/26/1985,5:30:14,千葉県東方沖,35°30.0'N,141°19.6'E,55,4.2,震度１,
12/25/1985,17:34:33,茨城県沖,36°37.7'N,141°11.9'E,47,4.3,震度１,
```

C. Python Code for CNN-GRU Model

https://github.com/3m0r9/EVALUATING-DEEP-LEARNING-MODELS-FOR-EARTHQUAKE-MAGNITUDE-PREDICTION

[10]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, GRU, Dense, Flatten,␣
 ↪Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
```

[11]:
```
# Load preprocessed data
# Assuming X_bmkg and y_bmkg are already preprocessed
import numpy as np
X_bmkg = np.load("X_bmkg_cnn_gru.npy")
y_bmkg = np.load("y_bmkg_cnn_gru.npy", allow_pickle=True)

# Verify the shape
print(f"X_bmkg shape: {X_bmkg.shape}")
print(f"y_bmkg shape: {y_bmkg.shape}")
```

```
X_bmkg shape: (92878, 10, 10)
y_bmkg shape: (92878,)
```

[46]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, BatchNormalization,␣
 ↪GRU, Dropout, Dense
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

from tensorflow.keras.layers import LeakyReLU

def build_optimized_cnn_gru_model(input_shape):
    model = Sequential()

    # CNN Layers
    model.add(Conv1D(filters=64, kernel_size=3, padding='same',␣
 ↪input_shape=input_shape))
```

```
    model.add(LeakyReLU(alpha=0.1))  # LeakyReLU for better gradient flow
    model.add(BatchNormalization())
    model.add(MaxPooling1D(pool_size=2))

    model.add(Conv1D(filters=128, kernel_size=3, padding='same'))
    model.add(LeakyReLU(alpha=0.1))
    model.add(BatchNormalization())
    model.add(MaxPooling1D(pool_size=2))

    # GRU Layers
    model.add(GRU(units=192, return_sequences=True, kernel_regularizer=l2(0.
↪0001)))
    model.add(Dropout(0.3))
    model.add(GRU(units=128, return_sequences=False, kernel_regularizer=l2(0.
↪0001)))
    model.add(Dropout(0.3))

    # Dense Layers
    model.add(Dense(units=64))
    model.add(LeakyReLU(alpha=0.1))
    model.add(Dropout(0.4))
    model.add(Dense(units=1, activation='linear'))  # Regression Output

    # Compile the model
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse',␣
↪metrics=['mae'])

    return model
```

[47]:
```
# Build the model
input_shape = (10, 10)  # Adjust based on your data
model = build_optimized_cnn_gru_model(input_shape)
```

[48]:
```
from tensorflow.keras.callbacks import LearningRateScheduler
# Define Callbacks
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=15,  # Stop if no improvement after 10 epochs
    restore_best_weights=True
)
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,  # Reduce learning rate by half
    patience=5,  # After 5 epochs of no improvement
    min_lr=1e-6  # Minimum learning rate
)
# Define the learning rate scheduler function
```

```
def lr_schedule(epoch, lr):
    if epoch % 5 == 0 and epoch != 0:  # Decay every 5 epochs, excluding the␣
 ↪first epoch
        return lr * 0.95  # Multiply the current learning rate by 0.9
    return lr

# Create the LearningRateScheduler callback
lr_scheduler = LearningRateScheduler(lr_schedule, verbose=1)
```

[49]:
```
# Adjust the input shape to match the new data
input_shape = (X_bmkg.shape[1], X_bmkg.shape[2])  # (10, 10)

# Build the model
model = build_optimized_cnn_gru_model(input_shape)
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_13 (Conv1D)          (None, 10, 64)            1984

 leaky_re_lu_19 (LeakyReLU)  (None, 10, 64)            0

 batch_normalization_13 (Ba  (None, 10, 64)            256
 tchNormalization)

 max_pooling1d_13 (MaxPooli  (None, 5, 64)             0
 ng1D)

 conv1d_14 (Conv1D)          (None, 5, 128)            24704

 leaky_re_lu_20 (LeakyReLU)  (None, 5, 128)            0

 batch_normalization_14 (Ba  (None, 5, 128)            512
 tchNormalization)

 max_pooling1d_14 (MaxPooli  (None, 2, 128)            0
 ng1D)

 gru_12 (GRU)                (None, 2, 192)            185472

 dropout_18 (Dropout)        (None, 2, 192)            0

 gru_13 (GRU)                (None, 128)               123648

 dropout_19 (Dropout)        (None, 128)               0
```

```
 dense_12 (Dense)            (None, 64)                8256

 leaky_re_lu_21 (LeakyReLU)  (None, 64)                0

 dropout_20 (Dropout)        (None, 64)                0

 dense_13 (Dense)            (None, 1)                 65

=================================================================
Total params: 344897 (1.32 MB)
Trainable params: 344513 (1.31 MB)
Non-trainable params: 384 (1.50 KB)
_________________________________________________________________
```

[50]:
```
# Train the model
history = model.fit(
    X_bmkg, y_bmkg,
    validation_split=0.2,
    epochs=100,  # Increased to 100 to allow for more learning
    batch_size=64,
    callbacks=[early_stopping, reduce_lr, lr_scheduler],
)
```

```
Epoch 1: LearningRateScheduler setting learning rate to 0.0010000000474974513.
Epoch 1/100
1161/1161 [==============================] - 13s 10ms/step - loss: 1.0422 - mae:
0.7938 - val_loss: 4.9795 - val_mae: 1.8339 - lr: 0.0010

Epoch 2: LearningRateScheduler setting learning rate to 0.0010000000474974513.
Epoch 2/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.8601 - mae:
0.7313 - val_loss: 2.6816 - val_mae: 1.2480 - lr: 0.0010

Epoch 3: LearningRateScheduler setting learning rate to 0.0010000000474974513.
Epoch 3/100
1161/1161 [==============================] - 11s 10ms/step - loss: 0.8101 - mae:
0.7111 - val_loss: 0.8677 - val_mae: 0.7159 - lr: 0.0010

Epoch 4: LearningRateScheduler setting learning rate to 0.0010000000474974513.
Epoch 4/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.7572 - mae:
0.6883 - val_loss: 1.5000 - val_mae: 0.8594 - lr: 0.0010

Epoch 5: LearningRateScheduler setting learning rate to 0.0010000000474974513.
Epoch 5/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.7182 - mae:
```

```
0.6705 - val_loss: 0.6828 - val_mae: 0.6595 - lr: 0.0010

Epoch 6: LearningRateScheduler setting learning rate to 0.0009500000451225787.
Epoch 6/100
1161/1161 [==============================] - 11s 10ms/step - loss: 0.6932 - mae:
0.6601 - val_loss: 0.6964 - val_mae: 0.6719 - lr: 9.5000e-04

Epoch 7: LearningRateScheduler setting learning rate to 0.0009500000160187483.
Epoch 7/100
1161/1161 [==============================] - 11s 10ms/step - loss: 0.6787 - mae:
0.6542 - val_loss: 0.7576 - val_mae: 0.7093 - lr: 9.5000e-04

Epoch 8: LearningRateScheduler setting learning rate to 0.0009500000160187483.
Epoch 8/100
1161/1161 [==============================] - 11s 10ms/step - loss: 0.6659 - mae:
0.6482 - val_loss: 0.6501 - val_mae: 0.6441 - lr: 9.5000e-04

Epoch 9: LearningRateScheduler setting learning rate to 0.0009500000160187483.
Epoch 9/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.6536 - mae:
0.6426 - val_loss: 0.6938 - val_mae: 0.6739 - lr: 9.5000e-04

Epoch 10: LearningRateScheduler setting learning rate to 0.0009500000160187483.
Epoch 10/100
1161/1161 [==============================] - 11s 10ms/step - loss: 0.6456 - mae:
0.6402 - val_loss: 0.6517 - val_mae: 0.6414 - lr: 9.5000e-04

Epoch 11: LearningRateScheduler setting learning rate to 0.0009025000152178108.
Epoch 11/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.6362 - mae:
0.6338 - val_loss: 0.6484 - val_mae: 0.6435 - lr: 9.0250e-04

Epoch 12: LearningRateScheduler setting learning rate to 0.0009025000035762787.
Epoch 12/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.6317 - mae:
0.6328 - val_loss: 0.6752 - val_mae: 0.6644 - lr: 9.0250e-04

Epoch 13: LearningRateScheduler setting learning rate to 0.0009025000035762787.
Epoch 13/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.6276 - mae:
0.6311 - val_loss: 0.7553 - val_mae: 0.7107 - lr: 9.0250e-04

Epoch 14: LearningRateScheduler setting learning rate to 0.0009025000035762787.
Epoch 14/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.6226 - mae:
0.6282 - val_loss: 0.6492 - val_mae: 0.6454 - lr: 9.0250e-04

Epoch 15: LearningRateScheduler setting learning rate to 0.0009025000035762787.
```

```
Epoch 15/100
1161/1161 [==============================] - 11s 10ms/step - loss: 0.6207 - mae:
0.6278 - val_loss: 0.7111 - val_mae: 0.6840 - lr: 9.0250e-04

Epoch 16: LearningRateScheduler setting learning rate to 0.0008573750033974647.
Epoch 16/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.6155 - mae:
0.6246 - val_loss: 0.6785 - val_mae: 0.6647 - lr: 4.2869e-04

Epoch 17: LearningRateScheduler setting learning rate to 0.0004286874900572002.
Epoch 17/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.6065 - mae:
0.6195 - val_loss: 0.6442 - val_mae: 0.6403 - lr: 4.2869e-04

Epoch 18: LearningRateScheduler setting learning rate to 0.0004286874900572002.
Epoch 18/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.6016 - mae:
0.6176 - val_loss: 0.6584 - val_mae: 0.6549 - lr: 4.2869e-04

Epoch 19: LearningRateScheduler setting learning rate to 0.0004286874900572002.
Epoch 19/100
1161/1161 [==============================] - 11s 10ms/step - loss: 0.6008 - mae:
0.6171 - val_loss: 0.6496 - val_mae: 0.6482 - lr: 4.2869e-04

Epoch 20: LearningRateScheduler setting learning rate to 0.0004286874900572002.
Epoch 20/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.6002 - mae:
0.6164 - val_loss: 0.6494 - val_mae: 0.6491 - lr: 4.2869e-04

Epoch 21: LearningRateScheduler setting learning rate to 0.0004072531155543402.
Epoch 21/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.5974 - mae:
0.6153 - val_loss: 0.6653 - val_mae: 0.6578 - lr: 4.0725e-04

Epoch 22: LearningRateScheduler setting learning rate to 0.0004072531010100242496.
Epoch 22/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.5965 - mae:
0.6146 - val_loss: 0.6513 - val_mae: 0.6492 - lr: 2.0363e-04

Epoch 23: LearningRateScheduler setting learning rate to 0.00020362655050121248.
Epoch 23/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.5905 - mae:
0.6114 - val_loss: 0.6447 - val_mae: 0.6448 - lr: 2.0363e-04

Epoch 24: LearningRateScheduler setting learning rate to 0.00020362655050121248.
Epoch 24/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.5885 - mae:
0.6103 - val_loss: 0.6247 - val_mae: 0.6310 - lr: 2.0363e-04
```

```
Epoch 25: LearningRateScheduler setting learning rate to 0.00020362655050121248.
Epoch 25/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.5862 - mae:
0.6091 - val_loss: 0.6365 - val_mae: 0.6401 - lr: 2.0363e-04

Epoch 26: LearningRateScheduler setting learning rate to 0.00019344522297615185.
Epoch 26/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.5853 - mae:
0.6082 - val_loss: 0.6454 - val_mae: 0.6447 - lr: 1.9345e-04

Epoch 27: LearningRateScheduler setting learning rate to 0.00019344521570019424.
Epoch 27/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.5843 - mae:
0.6078 - val_loss: 0.6452 - val_mae: 0.6466 - lr: 1.9345e-04

Epoch 28: LearningRateScheduler setting learning rate to 0.00019344521570019424.
Epoch 28/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.5839 - mae:
0.6074 - val_loss: 0.6530 - val_mae: 0.6510 - lr: 1.9345e-04

Epoch 29: LearningRateScheduler setting learning rate to 0.00019344521570019424.
Epoch 29/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.5833 - mae:
0.6071 - val_loss: 0.6390 - val_mae: 0.6395 - lr: 9.6723e-05

Epoch 30: LearningRateScheduler setting learning rate to 9.672260785009712e-05.
Epoch 30/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.5806 - mae:
0.6058 - val_loss: 0.6309 - val_mae: 0.6362 - lr: 9.6723e-05

Epoch 31: LearningRateScheduler setting learning rate to 9.188647745759226e-05.
Epoch 31/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.5796 - mae:
0.6050 - val_loss: 0.6435 - val_mae: 0.6440 - lr: 9.1886e-05

Epoch 32: LearningRateScheduler setting learning rate to 9.188647527480498e-05.
Epoch 32/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.5777 - mae:
0.6038 - val_loss: 0.6461 - val_mae: 0.6459 - lr: 9.1886e-05

Epoch 33: LearningRateScheduler setting learning rate to 9.188647527480498e-05.
Epoch 33/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.5763 - mae:
0.6034 - val_loss: 0.6310 - val_mae: 0.6368 - lr: 9.1886e-05

Epoch 34: LearningRateScheduler setting learning rate to 9.188647527480498e-05.
Epoch 34/100
```

```
1161/1161 [==============================] - 11s 9ms/step - loss: 0.5769 - mae:
0.6029 - val_loss: 0.6298 - val_mae: 0.6369 - lr: 4.5943e-05

Epoch 35: LearningRateScheduler setting learning rate to 4.594323763740249e-05.
Epoch 35/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.5746 - mae:
0.6019 - val_loss: 0.6334 - val_mae: 0.6400 - lr: 4.5943e-05

Epoch 36: LearningRateScheduler setting learning rate to 4.364607575553236e-05.
Epoch 36/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.5737 - mae:
0.6015 - val_loss: 0.6296 - val_mae: 0.6371 - lr: 4.3646e-05

Epoch 37: LearningRateScheduler setting learning rate to 4.3646075937431306e-05.
Epoch 37/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.5735 - mae:
0.6013 - val_loss: 0.6309 - val_mae: 0.6363 - lr: 4.3646e-05

Epoch 38: LearningRateScheduler setting learning rate to 4.3646075937431306e-05.
Epoch 38/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.5730 - mae:
0.6013 - val_loss: 0.6312 - val_mae: 0.6363 - lr: 4.3646e-05

Epoch 39: LearningRateScheduler setting learning rate to 4.3646075937431306e-05.
Epoch 39/100
1161/1161 [==============================] - 11s 9ms/step - loss: 0.5723 - mae:
0.6010 - val_loss: 0.6345 - val_mae: 0.6386 - lr: 2.1823e-05
```

[51]:
```
# report the model performance
loss, mae = model.evaluate(X_bmkg, y_bmkg, verbose=0)
print(f"Training Loss: {loss:.4f}")
print(f"Training MAE: {mae:.4f}")
```

```
Training Loss: 0.6030
Training MAE: 0.6163
```

[45]:
```
# Evaluate the model
training_loss, training_mae = model.evaluate(X_bmkg, y_bmkg, verbose=0)
print(f"Training Loss: {training_loss:.4f}")
print(f"Training MAE: {training_mae:.4f}")
```

```
Training Loss: 0.5936
Training MAE: 0.6116
```

[28]:
```
# Save the model
model.save('cnn_gru_bmkg_model2.h5')
```

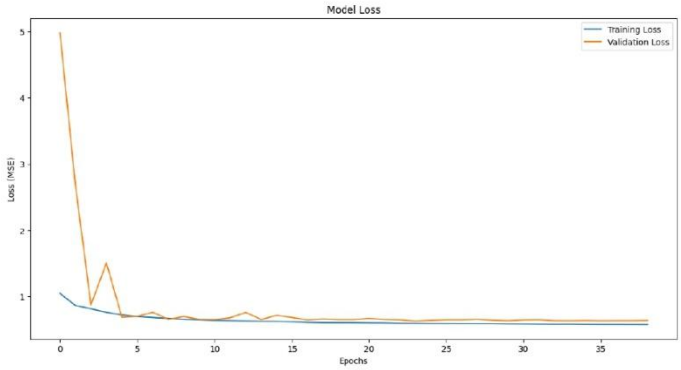
[36]:
```
# Plot training history
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 12))
```

[36]:
```
<Figure size 1200x1200 with 0 Axes>
```

```
<Figure size 1200x1200 with 0 Axes>
```

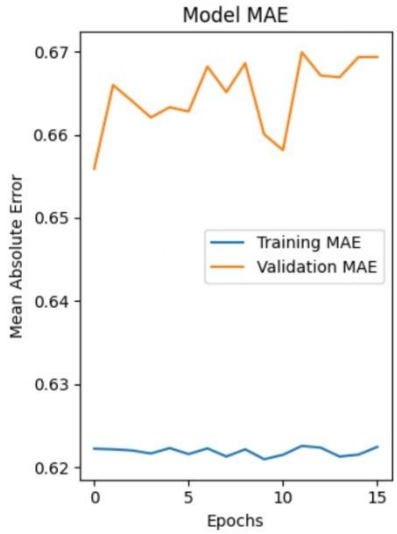
[53]:
```
# Plot loss
plt.figure(figsize=(14, 7))
#plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss (MSE)')
plt.legend()
```

[53]:
```
<matplotlib.legend.Legend at 0x2450aefd580>
```

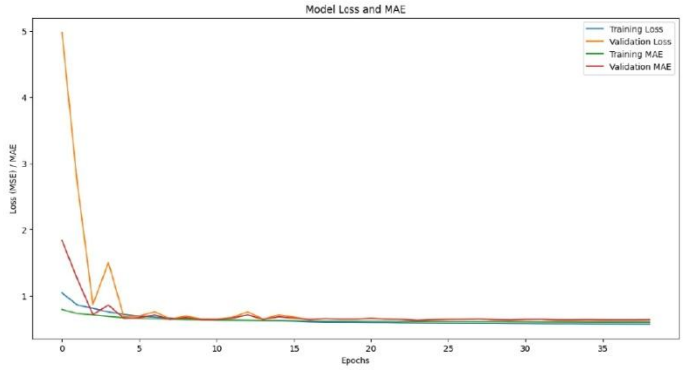


[38]:
```
# Plot MAE
plt.subplot(1, 2, 2)
plt.plot(history.history['mae'], label='Training MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.title('Model MAE')
plt.xlabel('Epochs')
plt.ylabel('Mean Absolute Error')
```

```
plt.legend()

plt.tight_layout()
plt.show()
```



[54]:
```
# Plot MAE and Loss together for better comparison
plt.figure(figsize=(14, 7))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.plot(history.history['mae'], label='Training MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.title('Model Loss and MAE')
plt.xlabel('Epochs')
plt.ylabel('Loss (MSE) / MAE')
plt.legend()

plt.show()
```


Model Loss and MAE
Training Loss
Validation Loss
Training MAE
Validation MAE
Loss (MSE) / MAE
Epochs
0
5
10
15
20
25
30
35
1
2
3
4
5

[ ]:

D. Python Code for CNN-BiLSTM Model with Attention

https://github.com/3m0r9/EVALUATING-DEEP-LEARNING-MODELS-FOR-EARTHQUAKE-MAGNITUDE-PREDICTION

[1]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, GRU, Dense, Flatten,␣
↪Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
```

[2]:
```
# Load preprocessed data
# Assuming X_jma and y_jma are already preprocessed
import numpy as np
X_jma = np.load("X_jma_cnn_bilstm.npy")
y_jma = np.load("y_jma_cnn_bilstm.npy", allow_pickle=True)

# Verify the shape
print(f"X_jma shape: {X_jma.shape}")
print(f"y_jma shape: {y_jma.shape}")
```

```
X_jma shape: (85330, 5, 10)
y_jma shape: (85330, 1)
```

[17]:
```
from sklearn.model_selection import train_test_split

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_jma, y_jma, test_size=0.
↪2, random_state=42)
```

[18]:
```
print(f"X_test shape: {X_test.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
X_test shape: (17066, 5, 10)
y_test shape: (17066, 1)
```

[20]:
```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import (Input, Conv1D, MaxPooling1D,␣
↪BatchNormalization,
                                     Bidirectional, LSTM, Dense, Dropout,␣
↪Attention)
```

```
# Input layer
input_layer = Input(shape=(X_jma.shape[1], X_jma.shape[2]))  # (timesteps,␣
 ↪features)

# Feature Extraction Block
conv1 = Conv1D(filters=16, kernel_size=5, strides=1, activation='relu',␣
 ↪padding='same')(input_layer)
bn1 = BatchNormalization()(conv1)
pool1 = MaxPooling1D(pool_size=2, strides=2, padding='same')(bn1)

conv2 = Conv1D(filters=32, kernel_size=3, strides=1, activation='relu',␣
 ↪padding='same')(pool1)
bn2 = BatchNormalization()(conv2)
pool2 = MaxPooling1D(pool_size=2, strides=2, padding='same')(bn2)

conv3 = Conv1D(filters=64, kernel_size=3, strides=1, activation='relu',␣
 ↪padding='same')(pool2)
bn3 = BatchNormalization()(conv3)
pool3 = MaxPooling1D(pool_size=2, strides=2, padding='same')(bn3)

conv4 = Conv1D(filters=128, kernel_size=3, strides=1, activation='relu',␣
 ↪padding='same')(pool3)
bn4 = BatchNormalization()(conv4)
pool4 = MaxPooling1D(pool_size=2, strides=2, padding='same')(bn4)

# Reshape is not required as pool4's output shape is already compatible with␣
 ↪LSTM
# Sequence Learning Block
lstm1 = Bidirectional(LSTM(units=128, return_sequences=True))(pool4)
dropout1 = Dropout(rate=0.2)(lstm1)
lstm2 = Bidirectional(LSTM(units=64, return_sequences=True))(dropout1)
dropout2 = Dropout(rate=0.2)(lstm2)

# Attention Mechanism
attention_output = Attention()([dropout2, dropout2])  # Query = Key = dropout2

# Fully Connected Prediction Block
fc1 = Dense(units=32, activation='relu')(attention_output)
fc2 = Dense(units=10, activation='relu')(fc1)
output_layer = Dense(units=1, activation='linear')(fc2)

# Define the model
model = Model(inputs=input_layer, outputs=output_layer)
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

[21]:
```
# Display the model summary
model.summary()
```

```
Model: "model_1"
__________________________________________________________________________________
_________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================
=================
 input_4 (InputLayer)        [(None, 5, 10)]           0         []

 conv1d_12 (Conv1D)          (None, 5, 16)             816
['input_4[0][0]']

 batch_normalization_12 (Ba  (None, 5, 16)             64
['conv1d_12[0][0]']
 tchNormalization)

 max_pooling1d_12 (MaxPooli  (None, 3, 16)             0
['batch_normalization_12[0][0]
 ng1D)                                                          ']

 conv1d_13 (Conv1D)          (None, 3, 32)             1568
['max_pooling1d_12[0][0]']

 batch_normalization_13 (Ba  (None, 3, 32)             128
['conv1d_13[0][0]']
 tchNormalization)

 max_pooling1d_13 (MaxPooli  (None, 2, 32)             0
['batch_normalization_13[0][0]
 ng1D)                                                          ']

 conv1d_14 (Conv1D)          (None, 2, 64)             6208
['max_pooling1d_13[0][0]']

 batch_normalization_14 (Ba  (None, 2, 64)             256
['conv1d_14[0][0]']
 tchNormalization)

 max_pooling1d_14 (MaxPooli  (None, 1, 64)             0
['batch_normalization_14[0][0]
 ng1D)                                                          ']

 conv1d_15 (Conv1D)          (None, 1, 128)            24704
['max_pooling1d_14[0][0]']
```

```
 batch_normalization_15 (Ba  (None, 1, 128)              512
['conv1d_15[0][0]']
 tchNormalization)

 max_pooling1d_15 (MaxPooli  (None, 1, 128)              0
['batch_normalization_15[0][0]
 ng1D)                                                              ']

 bidirectional_3 (Bidirecti  (None, 1, 256)              263168
['max_pooling1d_15[0][0]']
 onal)

 dropout_2 (Dropout)         (None, 1, 256)              0
['bidirectional_3[0][0]']

 bidirectional_4 (Bidirecti  (None, 1, 128)              164352
['dropout_2[0][0]']
 onal)

 dropout_3 (Dropout)         (None, 1, 128)              0
['bidirectional_4[0][0]']

 attention_1 (Attention)     (None, 1, 128)              0
['dropout_3[0][0]',
'dropout_3[0][0]']

 dense_3 (Dense)             (None, 1, 32)               4128
['attention_1[0][0]']

 dense_4 (Dense)             (None, 1, 10)               330
['dense_3[0][0]']

 dense_5 (Dense)             (None, 1, 1)                11
['dense_4[0][0]']

==================================================================================
=================
Total params: 466245 (1.78 MB)
Trainable params: 465765 (1.78 MB)
Non-trainable params: 480 (1.88 KB)
----------------------------------------------------------------------------------
-----------------
```

[22]:
```
# Callbacks
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
early_stopping = EarlyStopping(monitor='val_loss',
                               patience=10,
```

```
                                restore_best_weights=True)


reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                              factor=0.1,
                              patience=5,
                              min_lr=1e-6)

# Train the model
history = model.fit(X_jma, y_jma,
                    batch_size=64,
                    epochs=100,
                    validation_split=0.2,
                    callbacks=[early_stopping])
```

```
Epoch 1/100
1067/1067 [==============================] - 13s 8ms/step - loss: 0.0114 - mae:
0.0809 - val_loss: 0.0082 - val_mae: 0.0720
Epoch 2/100
1067/1067 [==============================] - 7s 7ms/step - loss: 0.0073 - mae:
0.0651 - val_loss: 0.0085 - val_mae: 0.0745
Epoch 3/100
1067/1067 [==============================] - 7s 7ms/step - loss: 0.0065 - mae:
0.0613 - val_loss: 0.0067 - val_mae: 0.0640
Epoch 4/100
1067/1067 [==============================] - 7s 7ms/step - loss: 0.0062 - mae:
0.0596 - val_loss: 0.0060 - val_mae: 0.0591
Epoch 5/100
1067/1067 [==============================] - 7s 7ms/step - loss: 0.0060 - mae:
0.0586 - val_loss: 0.0057 - val_mae: 0.0569
Epoch 6/100
1067/1067 [==============================] - 8s 7ms/step - loss: 0.0059 - mae:
0.0581 - val_loss: 0.0057 - val_mae: 0.0582
Epoch 7/100
1067/1067 [==============================] - 8s 7ms/step - loss: 0.0058 - mae:
0.0577 - val_loss: 0.0069 - val_mae: 0.0662
Epoch 8/100
1067/1067 [==============================] - 8s 7ms/step - loss: 0.0058 - mae:
0.0574 - val_loss: 0.0063 - val_mae: 0.0623
Epoch 9/100
1067/1067 [==============================] - 8s 7ms/step - loss: 0.0057 - mae:
0.0571 - val_loss: 0.0060 - val_mae: 0.0599
Epoch 10/100
1067/1067 [==============================] - 8s 7ms/step - loss: 0.0056 - mae:
0.0567 - val_loss: 0.0054 - val_mae: 0.0552
Epoch 11/100
1067/1067 [==============================] - 7s 7ms/step - loss: 0.0056 - mae:
```

```
0.0567 - val_loss: 0.0055 - val_mae: 0.0557
Epoch 12/100
1067/1067 [==============================] - 8s 7ms/step - loss: 0.0056 - mae:
0.0564 - val_loss: 0.0058 - val_mae: 0.0589
Epoch 13/100
1067/1067 [==============================] - 8s 7ms/step - loss: 0.0055 - mae:
0.0561 - val_loss: 0.0056 - val_mae: 0.0566
Epoch 14/100
1067/1067 [==============================] - 8s 7ms/step - loss: 0.0055 - mae:
0.0558 - val_loss: 0.0069 - val_mae: 0.0660
Epoch 15/100
1067/1067 [==============================] - 8s 7ms/step - loss: 0.0054 - mae:
0.0557 - val_loss: 0.0056 - val_mae: 0.0561
Epoch 16/100
1067/1067 [==============================] - 8s 7ms/step - loss: 0.0054 - mae:
0.0553 - val_loss: 0.0057 - val_mae: 0.0573
Epoch 17/100
1067/1067 [==============================] - 8s 7ms/step - loss: 0.0053 - mae:
0.0552 - val_loss: 0.0056 - val_mae: 0.0561
Epoch 18/100
1067/1067 [==============================] - 8s 7ms/step - loss: 0.0053 - mae:
0.0550 - val_loss: 0.0056 - val_mae: 0.0556
Epoch 19/100
1067/1067 [==============================] - 8s 7ms/step - loss: 0.0053 - mae:
0.0549 - val_loss: 0.0068 - val_mae: 0.0652
Epoch 20/100
1067/1067 [==============================] - 7s 7ms/step - loss: 0.0052 - mae:
0.0547 - val_loss: 0.0055 - val_mae: 0.0555
```

[23]:
```python
test_loss, test_mae = model.evaluate(X_test, y_test, verbose=1)
print(f"Test Loss (MSE): {test_loss}")
print(f"Test MAE: {test_mae}")
```

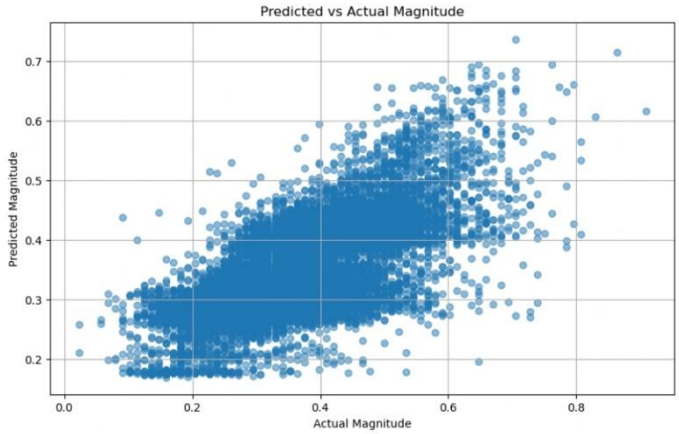
```
534/534 [==============================] - 1s 2ms/step - loss: 0.0055 - mae:
0.0554
Test Loss (MSE): 0.005500436760485172
Test MAE: 0.055425096303224564
```

[41]:
```python
from sklearn.metrics import mean_squared_error
import numpy as np

# Predict on test data
y_pred = model.predict(X_test)

# Reshape predictions and ground truth for compatibility
y_test_flat = y_test.flatten()  # Flatten if y_test has an extra dimension
y_pred_flat = y_pred.flatten()  # Flatten if y_pred has an extra dimension
```

```
# Calculate RMSE
mse = mean_squared_error(y_test_flat, y_pred_flat)
rmse = np.sqrt(mse)

print(f"Test RMSE: {rmse}")

# R score
from sklearn.metrics import r2_score

r2 = r2_score(y_test_flat, y_pred_flat)
print(f"R^2 Score: {r2}")
```

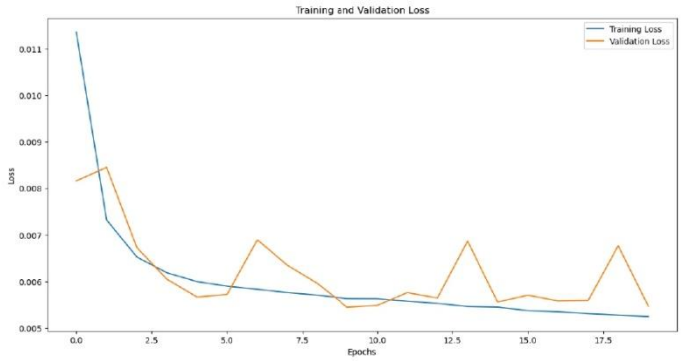
```
534/534 [==============================] - 1s 1ms/step
Test RMSE: 0.07416494240247815
R^2 Score: 0.5015814295333142
```

[34]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel("Actual Magnitude")
plt.ylabel("Predicted Magnitude")
plt.title("Predicted vs Actual Magnitude")
plt.grid()
plt.show()
```

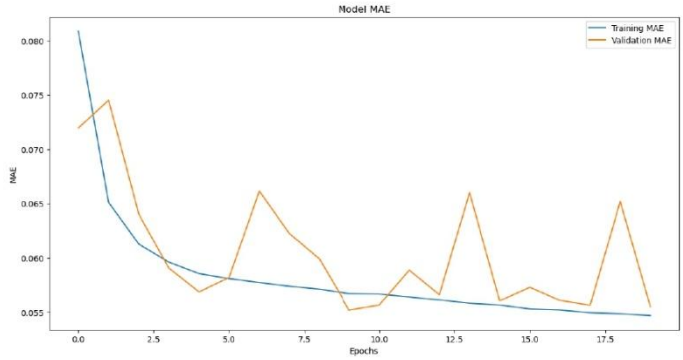


[ ]:

[35]:
```
# Plot the training history
plt.figure(figsize=(14, 7))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()
```



[42]:
```
# Plot Training and Validation MAE
plt.figure(figsize=(14, 7))
plt.plot(history.history['mae'], label='Training MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.title('Model MAE')
plt.xlabel('Epochs')
plt.ylabel('MAE')
plt.legend()
plt.show()
```



[37]:
```
# Save the model
model.save("cnn_bilstm_jma.h5")
```

```
C:\ProgramData\anaconda3\envs\skripsi\lib\site-
packages\keras\src\engine\training.py:3079: UserWarning: You are saving your
model as an HDF5 file via `model.save()`. This file format is considered legacy.
We recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(
```


E. Hyperparameter Tuning Details

https://github.com/3m0r9/EVALUATING-DEEP-LEARNING-MODELS-FOR-EARTHQUAKE-MAGNITUDE-PREDICTION