A comparative analysis   
of different deep learning models   
in fall detection on SmartWatch Dataset

|  |  |
| --- | --- |
| Thi Thuy Tien Tran  *Department of Computer Science*  *Lakehead University*  Thunder Bay, Ontario, Canada  ttran15@lakeheadu.ca | Azeem Abdul  *Department of Computer Science*  *Lakehead University*  Thunder Bay, Ontario, Canada  aabdul13@lakeheadu.ca |
| Darcus Angeline Peter  *Department of Computer Science*  *Lakehead University*  Thunder Bay, Ontario, Canada  dpeter3@lakeheadu.ca | Nimasha Warnakulasuriya  *Department of Computer Science*  *Lakehead University*  Thunder Bay, Ontario, Canada  nwarna@lakeheadu.ca |

*Abstract*— According to the World Health Organization, falls in older people are one of the significant health issues worldwide. Every year, it is estimated that 28-35% of people aged 65 years and older fall. The coming of IoT technology has eases the development of fall detection systems; it is thus a very promising solution for monitoring and detecting falls. This paper shall contrast the very latest transformer models with traditional deep-learning models by pitting state-of-the-art transformer models, such as Temporal Fusion Transformer and Bidirectional Encoder Representations from Transformers, against traditional deep learning models like Gated Recurrent Unit and Long Short-Term Memory for fall detection on the SmartFall dataset.

In the preprocessing of the data, the window size was 40 points to precisely capture the events of falls and thus correctly distinguish them from other activities. downsampling of the non-fall data was done to avoid significant class imbalance between fall and non-fall instances. The models were trained with 20% of the data reserved for validation with a learning rate of 0.0001, batch size of 32, and 50 epochs.

Results returned show that, compared to other deep learning models, BERT performs poorly in time series data. Among the models tested, the TFT model exhibited the best performance in the experiments, getting an accuracy of about 96%. In terms of precision, recall, and F1-score, TFT came first, which states it as the most effective model for fall detection in this study.

*Index Terms*—fall detection, deep learning, BERT, LSTM, GRU, TFT, SmartWatch dataset

# INTRODUCTION

For elderly people worldwide, fall has become a prevalent issue. The World Health Organization (WHO) reports that 28-35 percent of adults 65 years of age and older fall every year [1]. 30% to 50% of residents in long-term care facilities experience falls each year, with many recurrences [2]. The

incidence of falls and their negative health effects rise with age in humans [3]. Falls are a major contributing factor from

minor to severe injuries and even death [2]. Approximately two-thirds of all recorded injury-related deaths in individuals over 85 were attributed to falls [4]. About

20% of falls cause fatal injuries such head trauma, hip fractures, etc. [4]. Unintentional injuries brought on by falls is among the most expensive medical procedures [5]. In the US, fall-related hospitalizations cost an average of $17,483 per hospital stay for people 65 and older in 2004 [2]. By 2040, that number is expected to rise to $240 billion [2].

Internet of Things (IoT) is referred to a network of devices that are connected to the internet [6]. Applications IoT in healthcare can improve patient care, reduce expenses, increase access to care [6]. As IoT develops rapidly, using fall detection systems to track everyday activities and identify falls in the elderly has become more promising [7]. Early intervention can potentially reduce healthcare expenses and minimize the demand on caregivers [8]. While fall prevention is not the exclusive objective of a fall detection system, caregivers and medical professionals can benefit greatly from the information it offers [9].

Despite the increasing research on fall detection, there is a lack of studies applying transformers to fall datasets. The key contribution of this research is to conduct a comparative analysis of state-of-the-art transformer models (TFT and BERT) versus traditional deep learning models (GRU and LSTM) to evaluate their performance in developing fall detection models using SmartWatch datasets. The code for this study is available in this Github [repository](https://github.com/ttran15/fall_detection).

The remainder structure is arranged as follows: Section II examines related works; Section III provides information about the methodology; Section IV discusses results; and Section VI summarizes the findings.

# RELATED WORKS

The development of modern technologies for the detection and prevention of falls in older people has accomplished significant advances recently [8].

There are three types of fall detection studies that are currently being conducted: wearable sensors that monitor posture, computer vision systems that use video to identify falls, and environmental sensors that detect falls through vibrations or audio [10].

Regarding wearable sensors, a crucial sensor for wristwatch fall detection is accelerometer, which measures changes in an object's acceleration, the rate at which velocity changes over time [11]. In order to examine the effects of sensor placement and algorithm selection, a study on fall detection systems assessed a number of sensors using four machine learning algorithms (SVM, NB, Decision Tree, and KNN) [12]. The study concluded that the most effective sensors are those that are close to the body's center of gravity, which is the waist and chest, using ANOVA analysis [12]. Using three datasets (Smartwatch, Notch, and Farseeing), the researchers tested traditional (SVM, Naive Bayes) and non-traditional (Deep Learning) algorithms for fall detection and found that Deep Learning models consistently outperformed traditional ones [1].

Regarding environmental sensors, since wearable systems have a smaller sensing range and are more susceptible to interference from the environment, environmental fall detection methods—which use features like floor vibrations, sound, or infrared images—are more expensive and less accurate than wearable systems [10]. For emergency notifications, a system distinguishes falls from similar events is integrated with smart devices such as Google Home or Amazon Alexa [13]. It outperforms other methods with a 95.14% fall detection rate and a 97.22% accuracy [13]. Carla Taramasco et al. suggested classifying falls and notifying caregivers using very-low-resolution thermal sensors [14]. The data are then analyzed by three recurrent neural networks including LSTM, GRU, and Bi-LSTM [14].

Computer-vision fall detection uses probabilistic models and image processing to classify movements. While it doesn't require expensive equipment, it has drawbacks such as interference from the environment and privacy issues [10]. In contrast, skeleton recognition provides greater privacy protection and environmental resilience than human contour recognition [10]. Ahmad Lotfi et al. proposed method uses a multilayer perceptron neural network for classification, generates 10 features for fall detection, and extracts motion information around the body using an ellipse and bounding box [15]. With the UR Fall Detection dataset, it achieves a 99.60% detection rate and a 2.62% false alarm rate [15].

Since accelerometers are common in smart devices and falls cause a noticeable change in acceleration, this data is crucial for detecting falls [11]. Regarding models, traditional machine learning algorithms have the drawback of requiring human feature extraction and selection prior to learning, meaning that the efficacy of the algorithms depends on the researcher's ability to specify relevant features [1]. On the other hand, deep learning techniques provide a wide range of hyperparameter tuning options, which helps them overcome the limitations of traditional machine learning techniques [3]. Therefore, this study proposes a fall detection framework based on wearable-sensor fall detection dataset using deep learning techniques.

# METHODOLOGY

## Dataset

The dataset used in this research is Smart Watch dataset. SmartWatch dataset collected data from seven healthy participants who wore MS Band watches and engaged in simulated falls and activities of daily living (ADLs) like running and hand waving [1]. Four types of falls were performed onto a mattress, with ten repetitions of each direction [1]. Many sampling rates were used to collect the data, with testing determining that 31.25 Hz was the best option [1]. Real-time data labeling as "Fall" or "NotFall" was accomplished through a smartphone app; however, human labeling resulted in some errors that were later fixed by post-processing to guarantee accurate labeling of fall phases [1]. SmartWatch dataset can be accessed via this [link](https://userweb.cs.txstate.edu/~hn12/data/SmartFallDataSet/SmartWatch/).

SmartWatch dataset consists of two files, training dataset and testing dataset. Each file includes four columns: accelerometer\_x, accelerometer\_y, accelerometer\_z, and outcome. Outcome has two values "Fall" and "Not Fall". Table 1 summarize number of records in SmartWatch original training and testing dataset.

TABLE I

SmartWatch original dataset description

|  |  |  |  |
| --- | --- | --- | --- |
|  | Fall | Not Fall | Sum |
| Training | 4,550 | 29,470 | 34,020 |
| Testing | 2,275 | 14,954 | 17,229 |
| Sum | 6,825 | 44,424 | 51,249 |

Figure 1 illustrates signal values from the SmartWatch dataset, showing accelerometer measurements of daily body motion in the three axes: x, y, and z coordinates.

A graph showing a graph of a graph

Description automatically generated with medium confidence

Figure 1. Signal value extracted from SmartWatch dataset

## Techniques

Deep learning has demonstrated remarkable efficacy in applications related to computer vision, speech recognition, and natural language processing [16]. In this research, we use various kinds of deep learning models including LSTM, GRU, BERT, and TFT.

### LSTM

Long Short-term Memory (LSTM), a type of a recurrent neural network (RNN), is particularly good at learning long-term dependencies and is suitable for sequential data such as time-series [17]. In order to prevent the vanishing gradient problem, in which the gradient getting too small or large during backpropagation, they use input, forget, and output gates to controlling data flow and maintaining crucial information [17].

### GRU

Similar to LSTM, Gated Recurrent Unit (GRU) is a type of Recurrent Neural Network, but it has a simpler structure with no output gate [17]. Instead, it uses update and reset gates to choose which data should be passed to the output [17].

### BERT

BERT (Bidirectional Encoder Representations from Transformers) is a cutting-edge natural language processing method released by Google in 2018 [18]. The transformer architecture it uses has 12 encoder layers, 8 attention heads, 768 hidden units, 512 hidden units, and 110 million parameters in total [18].

We have implemented fall detection system using the DistilBERT transformer model for sequence classification. The initial steps involve loading the datasets, which consist of accelerometer readings. The data is saved in .npy format and is loaded using the load\_np\_array function. This function retrieves the feature arrays (X\_train\_fall, X\_train\_notfall, X\_test\_fall, and X\_test\_notfall) and their corresponding label arrays. The fall and not-fall datasets are then concatenated to form comprehensive training and testing datasets, ensuring the model has a balanced view of both types of events.

To prepare the data for the DistilBERT model, the numerical accelerometer readings are converted into strings using the convert\_to\_string function. This step is crucial because DistilBERT, being a text-based model, requires textual input. The data is further split into training and validation sets to monitor the model's performance during training. The DistilBERT tokenizer is then used to tokenize the textual data, converting it into a format suitable for the model.

A custom dataset class, FallDetectionDataset, is defined to handle the tokenized data. This class utilizes the tokenizer to encode the text data, ensuring it fits the model's expected input format. Instances of this dataset class are created for the training, validation, and testing datasets. DataLoader instances are also created to handle batching and shuffling of data during training and evaluation.

The training process is managed using the Trainer class from the Hugging Face Transformers library. Training arguments are specified, including the number of epochs, batch size, learning rate, and strategies for logging and evaluation. An early stopping callback is also included to halt training if the model's performance does not improve, preventing overfitting.

During training, the model learns to classify the accelerometer readings as either a fall or not fall. After training, the model is evaluated on the validation dataset to select the best-performing model. The final evaluation is conducted on the test dataset, where various performance metrics such as accuracy, precision, recall, F1 score, and ROC AUC are computed.

The code also includes functions to visualize the model's performance. A confusion matrix is plotted to show the distribution of true and predicted labels, providing insights into the types of errors made by the model. Additionally, the ROC curve is plotted to illustrate the model's ability to distinguish between the two classes.

### TFT

Temporal Fusion Transformer (TFT) is a deep neural network based on attention for multi-horizon forecasting [19]. To integrate data from multiple time steps, TFT uses multi-head attention, LSTMs for local time-dependent processing, and Gated Residual Network blocks for effective information flow [19].

## Experimental Designs

Fall is an activity that happens over a period of time, so it is important to figure out how many steps in the past the network should be trained on [1]. The network won't fully capture the scope of the activity if it is trained on a small number of previous steps [1]. On the other hand, the network might consider irrelevant historical accelerometer data if we train it over an excessive number of steps [1].

We set a 40-point window size create data instances from training and testing dataset. This is sufficient time to record the features of a fall and exclude certain actions that could be considered falls [1]. Additionally, we labeled the data instances that were generated by assigning a fall label to each instance that had 25 or more fall label signal points within the window size [20]. If not, the data instance was classified as non-fall [20]. The generated training and testing datasets are displayed in Tables II.

TABLE II

SmartWatch generated dataset description

|  |  |  |  |
| --- | --- | --- | --- |
|  | Fall | Not Fall | Sum |
| Training | 2,912 | 31,068 | 33,980 |
| Testing | 1,456 | 15,733 | 17,189 |
| Sum | 4,368 | 46,801 | 51,169 |

As shown in Table II, the datasets used in the study were highly imbalanced, with significantly more non-fall instances compared to fall instances. Unbalanced class distribution could lead to poor classification results because the deep learning method is skewed or biased toward the dominant class rather than the minority class [20]. To address this, the non-fall data was downsampled to match the number of fall instances in both generated training and testing dataset. As a result, we randomly selected 2,912 non-fall instances out of 31,068 in generated training dataset, and 1,456 non-fall instances of 15,733 in generated testing dataset. The final training and testing dataset are illustrated in Table III.

TABLE III

SmartWatch final dataset description

|  |  |  |  |
| --- | --- | --- | --- |
|  | Fall | Not Fall | Sum |
| Training | 2,912 | 2,912 | 5,824 |
| Testing | 1,456 | 1,456 | 2,912 |
| Sum | 4,368 | 4,368 | 8,736 |

The generated dataset was used to train and evaluate the performance of four different models: LSTM (Long Short-Term Memory), GRU (Gated Recurrent Unit), BERT (Bidirectional Encoder Representations from Transformers), and TFT (Temporal Fusion Transformer).

* The LSTM model architecture consists of a LSTM layer, dense layer, dropout layer and output layer. The input to the model is resized into a form that resembles the training data’s dimensions. LSTM layer is set with 256 as the number of units and input shapes are provided which enables the capacity to learn pattern and dependencies in the data. After the LSTM layer, we added a dense layer with 128 neurons and passed it through a ReLU activation function. This layer is benefical in bringing nonlinearity to the model and thus, such nonlinear patterns from the data can be learned by the model. Again, regularization technique called dropout layer was used with a dropout rate of 0.5. This will effectively drop out half the input units during training thus improving the model’s regularity and its performance. The last layer is a dense layer with one neuron and a sigmoid activation, which is used for binary classification, and it outputs a probability value that ranges from zero to one concerning the event of a fall.
* About GRU architecture, the input to the model is then reshaped according to the dimensions of the training data, representing sequence length and number of features respectively. We configured the GRU layer with 256 units. Then, we applied a fully connected Dense layer of 32 units and applied the ReLU activation function after the GRU layer. This layer helps in the addition of nonlinearity into the model, therefore allowing the model to pick complicated patterns from the data. To prevent overfitting, a dropout layer has been used with a dropout rate of 0.5. This will randomly set half of the input units to zero during training, helping the model to be more robust and generalizable. The last layer is a dense layer with one unit and sigmoid activation which is very appropriate for binary classification, as it returns a probability value between zero and one regarding the occurrence of a fall.
* Regarding BERT model, a custom dataset class, FallDetectionDataset, is created to handle the dataset's text and label fields. This class leverages a tokenizer to preprocess the text data, converting it into a format that BERT can process. The tokenizer ensures that the text sequences are appropriately padded or truncated to a specified maximum length. The DistilBERT model is then initialized for sequence classification tasks. This involves setting up the model to output two classes (fall and not fall). The model's architecture is specifically designed to handle classification tasks by leveraging the powerful contextual embeddings provided by BERT. The training phase involves setting up an optimizer to adjust the model's parameters. A training loop iterates over the dataset in batches, passing the tokenized inputs through the model to obtain predictions. The loss between the predicted labels and actual labels is calculated, and backpropagation is used to update the model's weights.
* A Temporal Fusion Transformer architecture using Long Short-Term Memory units in pursuit of our objective: modeling temporal dependencies that would improve the prediction accuracy. The model input shape was set to ⁠ (40, 3) ⁠, where the first value is the sequence length, and the latter is the number of features. So at the core lies an LSTM layer with 256 units. LSTMs are one kind of recurrent neural network that is particularly well known for its ability to handle long-term dependencies in sequential data and thus appropriate in the processing and prediction of time-series data. The LSTM layer outputs sequences, which are then treated by a Multi-Head Attention layer with four heads. This would increase the attention to different parts of the sequence, further enhancing this model's ability in learning. A gated mechanism is added to the model with a Dense layer of 256 units and the sigmoid activation step-through that will allow the model to have some form of control of information flow and help the model retain relevant features. Then, add a Dense layer with 32 units and ReLU activation to introduce non-linearity into the model, which enables it to learn complex patterns, and a Dropout Layer with a rate of 0.5 is added to prevent overfitting. The last output layer is a dense layer with one unit and sigmoid activation, which is suitable for binary classification because the output will be a value between zero and one, representing the likelihood of a fall.

For models’ parameters, we set a pre-defined learning rate with a value of 0.0001. Also, we used the binary cross-entropy loss function and optimized this model for binary classification. As an evaluation metric of model performance, we chose accuracy. 50 epochs of training were conducted with a batch size of 32. We used 20% for the validation split to check how well our model worked on data it had not seen before. The early stopping callback will stop training if the model fails to perform better in the validation set in order to prevent overfitting and save time during training. The models' performance is then assessed using the testing dataset. Model parameters are represented in Table IV.

TABLE IV

Model Parameters

|  |  |
| --- | --- |
|  | Value |
| learning rate | 0.0001 |
| batch size | 32 |
| validation split ratio | 0.2 |
| number of epochs | 50 |
| Early stopping | patience=1 |

## Evaluation Metrics

Accuracy, precision, recall, and roc are used to evaluate the performance of the models.

Accuracy is the percentage of correctly classified data [21].

(1)

where FN stands for false negative rate, FP for false positive rate, TN for true negative rate, and TP for true positive rate.

When a binary classification precision score is 100%, all items identified as belonging to the positive class actually do belong to the positive class [21].

(2)

Recall or sensitivity estimates the likelihood of accurately identifying positive cases from those who genuinely have the target condition [21].

(3)

F1-score is the harmonic mean of recall and precision [3].

(4)

where stands for the weighting coefficient. In case F1-score,

AUC stands for the degree or measure of separability, and ROC is a probability curve [13]. They provide an indication of the model's degree of class distinction [13]. An AUC that is close to 1 indicates that a model is good in terms of separability [13].

# RESULTS

The ROC is used to visualize the effectiveness of a model’s classification performance [13].

A graph of a function

Description automatically generated with medium confidenceA blue squares with white text

Description automatically generated

|  |  |
| --- | --- |
|  |  |
| ROC Curve | Confusion Matrix |

Figure 2. ROC Curve and Confusion Matrix of LSTM

LSTM-**Confusion Matrix:**

The model is better at correctly identifying ‘Not Fall’ instances (1395 true negatives) than ‘Fall’ instances (1331 true positives).

There is a notable number of false negatives (125), meaning many actual falls are being predicted as not falls.

There are also a few false positives (61), where the model predicts a fall when there is none.

**LSTM-ROC curve**

The orange curve represents the performance of the fall detection model.

The area under the curve (AUC) is 0.9361, which indicates the model has excellent performance.

The curve shows that the model has strong discriminative ability, meaning it can effectively distinguish between falls and not falls.

A graph of a function

Description automatically generated with medium confidenceA blue squares with white text

Description automatically generated

|  |  |
| --- | --- |
|  |  |
| ROC Curve | Confusion Matrix |

Figure 3. ROC Curve and Confusion Matrix of GRU

**GRU-Confusion Matrix:**

The model is better at correctly identifying ‘Not Fall’ instances (1386 true negatives) than ‘Fall’ instances (1393 true positives).

There is a relatively low number of false negatives (63), meaning many actual falls are being predicted correctly.

There are also a few false positives (70), where the model predicts a fall when there is none.

**GRU-ROC curve:**

The orange curve represents the performance of the fall detection model.

The area under the curve (AUC) is 0.9543, which indicates the model has excellent performance.

The curve shows that the model has strong discriminative ability, meaning it can effectively distinguish between falls and not falls.

A graph of a curve

Description automatically generatedA blue squares with white text

Description automatically generated

|  |  |
| --- | --- |
|  |  |
| ROC Curve | Confusion Matrix |

Figure 4. ROC Curve and Confusion Matrix of BERT

BERT: **Confusion Matrix**

The model is better at correctly identifying 'Not Fall' instances (1005 true negatives) than 'Fall' instances (736 true positives).

There is a significant number of false negatives (720), meaning many actual falls are being predicted as not falls.

There are also a considerable number of false positives (451), where the model predicts a fall when there is none.

BERT: **ROC Curve**

The blue curve represents the performance of the fall detection model.

The area under the curve (AUC) is 0.6476, which indicates the model has a better performance than a random classifier (AUC = 0.5) but still far from perfect (AUC = 1.0).

The curve shows that the model has moderate discriminative ability, meaning it can somewhat distinguish between falls and not falls but there is room for improvement.

TFT:

A graph of a function

Description automatically generated with medium confidenceA blue squares with white text

Description automatically generated

|  |  |
| --- | --- |
|  |  |
| ROC Curve | Confusion Matrix |

Figure 5. ROC Curve and Confusion Matrix of TFT

Figure 5 illustrates ROC curve and confusion matrix of TFT.

**TFT-Confusion Matrix:**

The model is better at correctly identifying ‘Not Fall’ instances (1404 true negatives) than ‘Fall’ instances (1395 true positives).

There is a relatively low number of false negatives (61), meaning many actual falls are being predicted correctly.

There are also a few false positives (52), where the model predicts a fall when there is none.

**TFT-ROC curve:**

The orange curve represents the performance of the fall detection model.

The area under the curve (AUC) is 0.9612, which indicates the model has excellent performance.

The curve shows that the model has strong discriminative ability, meaning it can effectively distinguish between falls and not falls.

TABLE V

Models’ performance comparison

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Evaluation Metrics | LSTM | GRU | BERT | TFT |
| Accuracy | 0.9361 | 0.9543 | 0.5979 | 0.9612 |
| Precision | 0.9562 | 0.9522 | 0.6013 | 0.9641 |
| Recall | 0.9141 | 0.9567 | 0.5979 | 0.9581 |
| F1-score | 0.9347 | 0.9544 | 0.5944 | 0.9611 |

The performance comparison of four models, evaluated using various metrics, is shown in Table V.

The evaluation metrics reflect that both LSTM and GRU perform very well on the accuracy, precision, recall, and F1-score. The LSTM model achieved an accuracy of 0.9361, a precision of 0.9562, a recall of 0.9141, and an F1-score of 0.9347, while GRU performed slightly better: accuracy of 0.9543, precision of 0.9522, recall of 0.9567, and F1-score of 0.9544. In contrast, BERT performed poorer with an accuracy of 0.5979, a precision of 0.6013, a recall of 0.5979, and an F1-score of 0.5944. This could be caused by the model being focused more on semantic understanding and less on sequential data processing. The TFT model competitively works with an accuracy of 0.9612, a precision of 0.9641, recall of 0.9581, and an F1-score of 0.9611, which has proved the effectiveness of this method in the integration of temporal and feature-based information about fall detection tasks. The best model on this dataset was provided by the TFT model, which means that this method is more capable of handling the complexities in the fall detection problem based on sensor data.

CONCLUSION

In summary, through this research, we have developed sophisticated deep learning models to evaluate the SmartWatch dataset, improving classification performance of fall detection. Furthermore, we conducted a methodical comparison of LSTM, GRU, BERT, TFT to assess each deep learning method's efficiency. As a result, we discover that that BERT is less effective for time series data compared to other deep learning models. Meanwhile, TFT achieved the best performance.

In the future, we would like to apply training set resampling techniques to address the imbalanced classes issue. Instead of depending just on one random downsampling technique, we will run multiple downsampling iterations, then calcualte the average result. This method aims to mitigate bias when selecting dataset for training, leading to increase the reliability of model.

# REFERENCES

[1] T. Mauldin, M. Canby, V. Metsis, A. Ngu, and C. Rivera, “SmartFall: A Smartwatch-Based Fall Detection System Using Deep Learning,” *Sensors*, vol. 18, no. 10, p. 3363, Oct. 2018, doi: https://doi.org/10.3390/s18103363.

[2] M. Musci, D. De Martini, N. Blago, T. Facchinetti, and M. Piastra, “Online Fall Detection Using Recurrent Neural Networks,” *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1276–1289, Jul. 2021, doi: <https://doi.org/10.1109/tetc.2020.3027454>.

[3] M. Farsi, “Application of ensemble RNN deep neural network to the fall detection through IoT environment,” *Alexandria Engineering Journal*, Jul. 2020, doi: <https://doi.org/10.1016/j.aej.2020.06.056>.

[4] Md. M. Islam et al., “Deep Learning Based Systems Developed for Fall Detection: A Review,” *IEEE Access*, vol. 8, pp. 166117–166137, 2020, doi: https://doi.org/10.1109/access.2020.3021943.

[5] Z. Jiang et al., "Fall Detection Systems for Internet of Medical Things Based on Wearable Sensors: A Review," *IEEE Internet of Things Journal*, doi: 10.1109/JIOT.2024.3421336.

[6] R. Kaur and R. Sharma, “Wearable sensors and datasets for evaluating systems predicting falls and activities of daily living: recent advances and methodology,” *Multimedia Tools and Applications*, Jun. 2024, doi: https://doi.org/10.1007/s11042-024-19504-1.

[7] X. Wu, Y. Zheng, C.-H. Chu, L. Cheng, and J. Kim, “Applying deep learning technology for automatic fall detection using mobile sensors,” *Biomedical Signal Processing and Control*, vol. 72, p. 103355, Feb. 2022, doi: https://doi.org/10.1016/j.bspc.2021.103355.

[8] O. Reyad, H. Shehata, and M. Karar, “Developed Fall Detection of Elderly Patients in Internet of Healthcare Things,” *Computers, Materials & Continua*, vol. 76, no. 2, pp. 1689–1700, 2023, doi: https://doi.org/10.32604/cmc.2023.039084.

[9] T. Theodoridis, V. Solachidis, N. Vretos, and P. Daras, “Human fall detection from acceleration measurements using a recurrent neural network,” *Precision Medicine Powered by pHealth and Connected Health*, pp. 145–149, Nov. 2017. doi:10.1007/978-981-10-7419-6\_25.

[10] C.-B. Lin, Z. Dong, W.-K. Kuan, and Y.-F. Huang, “A Framework for Fall Detection Based on OpenPose Skeleton and LSTM/GRU Models,” *Applied Sciences*, vol. 11, no. 1, p. 329, Dec. 2020, doi: https://doi.org/10.3390/app11010329.

[11] Nader Maray, Anne, J. Ni, M. Debnath, and L. Wang, “Transfer Learning on Small Datasets for Improved Fall Detection,” *Sensors*, vol. 23, no. 3, pp. 1105–1105, Jan. 2023, doi: https://doi.org/10.3390/s23031105.

[12] J. Santoyo-Ramón, E. Casilari, and J. Cano-García, “Analysis of a Smartphone-Based Architecture with Multiple Mobility Sensors for Fall Detection with Supervised Learning,” *Sensors*, vol. 18, no. 4, p. 1155, Apr. 2018, doi: https://doi.org/10.3390/s18041155.

[13] J. Clemente, F. Li, M. Valero, and W. Song, “Smart Seismic Sensing for Indoor Fall Detection, Location, and Notification,” *IEEE Journal of Biomedical and Health Informatics*, vol. 24, no. 2, pp. 524–532, Feb. 2020, doi: https://doi.org/10.1109/jbhi.2019.2907498.

[14] C. Taramasco et al., “A Novel Monitoring System for Fall Detection in Older People,” *IEEE Access*, vol. 6, pp. 43563–43574, 2018, doi: https://doi.org/10.1109/access.2018.2861331.

[15] A. Lotfi, S. Albawendi, H. Powell, K. Appiah, and C. Langensiepen, “Supporting Independent Living for Older Adults; Employing a Visual Based Fall Detection Through Analysing the Motion and Shape of the Human Body,” *IEEE Access*, vol. 6, pp. 70272–70282, 2018, doi: https://doi.org/10.1109/access.2018.2881237.

[16] T. R. Mauldin, Anne, Vangelis Metsis, M. E. Canby, and Jelena Tesic, “Experimentation and Analysis of Ensemble Deep Learning in IoT Applications,” *The Internet of Things*, vol. 5, no. 1, pp. 133–149, Jan. 2019.

[17] P. T. Yamak, L. Yujian, and P. K. Gadosey, “A comparison between Arima, LSTM, and GRU for time series forecasting,” *Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence*, Dec. 2019. doi:10.1145/3377713.3377722.

[18] H. Ramirez, S. A. Velastin, S. Cuellar, E. Fabregas, and G. Farias, “Bert for activity recognition using sequences of skeleton features and data augmentation with gan,” *Sensors*, vol. 23, no. 3, p. 1400, Jan. 2023. doi:10.3390/s23031400.

[19] B. Lim, S. Ö. Arık, N. Loeff, and T. Pfister, “Temporal Fusion Transformers for interpretable multi-horizon time series forecasting,” *International Journal of Forecasting*, vol. 37, no. 4, pp. 1748–1764, Oct. 2021, doi: https://doi.org/10.1016/j.ijforecast.2021.03.012.

[20] M. S. Al-Rakhami et al., “FallDeF5: A Fall Detection Framework Using 5G-Based Deep Gated Recurrent Unit Networks,” *IEEE Access*, vol. 9, pp. 94299–94308, 2021, doi: https://doi.org/10.1109/access.2021.3091838.

‌[21] G. Santos, P. Endo, K. Monteiro, E. Rocha, I. Silva, and T. Lynn, “Accelerometer-Based Human Fall Detection Using Convolutional Neural Networks,” *Sensors*, vol. 19, no. 7, p. 1644, Apr. 2019, doi: https://doi.org/10.3390/s19071644.

‌