POLITECNICO
MILANO 1863

Restoration of missing or low-quality 12-lead ECG signals using ensemble deep-learning model with optimal combination:

MATLAB and Python Implementation

Biomedical Signal Processing Laboratory (POLIMI 2024-2025)

Group number: 2

Jazmín Azul Fermani - Person code: 11072011 - jazminazul.fermani@mail.polimi.it
Federico Sangineto Cotrina - Person code: 11073413 - federico.sangineto@mail.polimi.it
Shahryar Namdari Ghareghani - Person code: 10959487 - shahryar.namdari@mail.polimi.it

24/01/2025

Table of content

1. Abstract

This report focuses on replicating the study titled *"Restoration of missing or low-quality 12-lead ECG signals using ensemble deep-learning model with optimal combination,"* published in *Biomedical Signal Processing and Control*. The original study addresses critical challenges in 12-lead electrocardiograms (ECGs), which are widely used to diagnose cardiac diseases. These challenges arise from low-quality signals caused by high-frequency and low-frequency noise as well as missing signals due to mechanical or operator errors. Such issues can lead to delays in diagnosis, increased medical costs, and diminished reliability of cardiac assessments. The study's objective was to develop an ensemble deep-learning model capable of restoring missing ECG signals with high accuracy, thereby enhancing the quality and usability of 12-lead ECG recordings in medical diagnostics.

In this report, we aim to replicate the methodology and findings of the original study. Detailed explanations of the methodology, dataset preparation, model implementation, and analysis will be provided in the following sections of this report. Through this replication, we hope to critically evaluate the study and further our understanding of advanced ECG restoration techniques.

2. Basic procedure

2.1. Original algorithm proposal

The algorithm proposed in the paper utilizes a hybrid approach, **combining statistical methods and deep-learning techniques**, to address the challenge of restoring missing ECG signals. Specifically, it identifies the optimal **combination of three electrocardiogram** (ECG) leads for each missing signal using **linear regression** (LR). This optimal combination serves as the input for a deep-learning model that integrates **convolutional neural networks** (CNN) and **bidirectional long short-term memory** (Bi-LSTM). The CNN extracts **spatial** features from the ECG signals, while the Bi-LSTM captures their **temporal** characteristics, enabling the model to effectively process the complex spatial-temporal dynamics inherent in long-term measurements.

To ensure data quality, ECG recordings **shorter than 10 seconds or with sampling rates deviating from the international standard of 500 Hz** were excluded from the dataset. Furthermore, to address noise issues stemming from the original XML database format, preprocessing steps were applied, including a **bandpass filter (0.05–150 Hz)** to remove high- and low-frequency noise and **asymmetrically reweighted penalized least squares smoothing** (arPLS) to eliminate baseline wander. These preprocessing techniques helped improve the signal quality for both the LR and deep-learning stages of the model.

The model trained on the optimal data combinations derived from LR was validated using 3-fold Monte Carlo cross-validation (MCCV). This method involves randomly splitting

the data into training and testing subsets three times, ensuring robust evaluation of model performance. MCCV is particularly well-suited for managing large datasets like the one used in the study, as it reduces the risk of overfitting while providing reliable performance metrics for the proposed restoration algorithm.

The authors' approach demonstrated significant improvements in the accuracy of restored signals, achieving an average root mean square error (RMSE) of 0.037 μV and a cosine similarity of 0.991, making the model suitable for both emergency medical systems and synthetic ECG dataset generation for research purposes.

2.2. Applied changes and reasoning

2.2.1. Dataset modifications and description

The first choice we had to make when implementing the proposed algorithm was that of choosing an alternative dataset, as the **KURIAS-ECG Database**[1] used in the original paper has been **unavailable since November 2021**, and consulting with Hyung Joon Joo (one of the researchers that worked on the paper) revealed that this was due to a legal issue regarding exporting personal health information abroad was raised for the project.

As such, we chose an **alternative**[2] that contains high-quality labeled 12-lead electrocardiogram (ECG) signals that were introduced by Jianwei Zheng, Hangyuan Guo, and Huimin Chu and developed collaboratively by Shaoxing People's Hospital, Ningbo First Hospital and Chapman university. It is a resource developed to support advancements in automatic arrhythmia and other conditions diagnosis using machine learning.

It contains data from 45,152 patients recorded at a frequency of 500 Hz representing various cardiovascular conditions, all labeled by professional experts. It includes common arrhythmias such as atrial fibrillation (AFIB), Sinus Bradycardia (SB), premature ventricular contraction (PVC), Atrial Tachycardia (AT), Atrial Premature Beat (APB), bundle branch blocks (LBBB, RBBB) as well as additional unspecified rhythms or abnormalities with the diagnostic codes mapped to SNOMED CT. Patient demographic attributes such as age and gender are also included.

The ECGs were resting recorded over a period of 10 seconds with subjects in resting position.

The recordings are structured hierarchically into two level folder directories, with each folder containing up to 100 ECG recordings. The data is available in WFDB format, comprising paired .mat and .hea files, along with annotation files in CSV format. The WFDB toolbox is designed to read .dat and .hea files so in order to use it the .mat files were first stored as .dat files using the load function in matlab.

[1] Yoo, H., et al.
[2] Zheng, J. et al.

The recordings are encoded with a 32-bit resolution, with a microvolt amplitude unit and a range of -32,768 to 32,767.

The dataset is suited for training and validating machine learning models as it contains large-size data that correspond to healthy subjects as well as subjects with different diagnoses.

2.2.2. Additional noise removal

When running our MATLAB preprocessing code we ran some peripheral functions to check our output signals' SNR, with which we **obtained negative values**, and so our initial threshold (SNR > 2.5 dB) for rejecting noisy signals rejected all of our signals. To compensate for this **we had to implement specifications other than the ones mentioned** by the authors.

To compensate for this, we **added an additional highpass filter with a cutoff at 0.5 Hz**; this helped us reach a SNR ~ 0 dB, which - while not yet a satisfying result - was an improvement.

However, we were unable to further improve our signal for some time, which led us to change our approach and **implement additional threshold values** apart from the SNR (this is detailed in point 3.2), this, **in combination with wavelet denoising** using Daubechies mother wavelets - whose shape is particularly fitting to the QRS waveforms' characteristic shape - allowed us to better our SNR and signal visualization.

In the end we ended up evaluating our signal's quality mainly on those thresholds, while not considering SNR, as it allowed us to maximize both our validated signals and said signals' quality.

2.2.3. Modification in model training methods

Repeated Splits: The paper evaluates each combination using a single split, whereas we perform **multiple splits and average the RMSE** for added robustness.

GPU Acceleration: We leverage **GPU libraries to speed up the large number of least-squares solutions required** when testing all three-lead subsets.

By integrating these modifications, our method follows the same linear regression framework in spirit but introduces additional stability through repeated splits and delivers faster computations with GPU resources.

3. Methodology and code description

3.1. Data extraction

In order to extract the data from the database the **WFDB Toolbox for MATLAB**[3] was downloaded, said toolbox is designed to read .dat and .hea files so the .mat files given in the dataset were first stored as .dat files using the load function in MATLAB.

The extracted data exceeds MATLAB's 2GB file size limit for standard .mat files, which use version 7.0 by default. For this reason, MAT-file version 7.3 was required to store the data in a matrix. This version allows storing variables larger than 2GB by saving data in HDF5 format that may have slower saving and loading times.

3.2. Preprocessing

After the raw data was extracted, it was saved in a matrix called *extracted_ecg_data.mat* and this matrix is loaded in the code. The overall process is twofold: first, **we filter our signals to maximize their usability**; second, **we remove outliers that maintain noisy properties despite the filtering**. Among the parameters for the following preprocessing some are defined such as: the *sampling frequency*, *band-pass* and *high-pass cut-off frequencies*, and two parameters, *lambda* and *max_iter* for the arPLS baseline correction method.

Since the data set is large, having a total of approximately 45,000 12-lead ECG signals, **these are processed in chunks**. Therefore, some parameters were defined like *chunk_size* and *num_chunks* (calculated by dividing the total number of records by a chosen chunk_size) which represents the number of records processed at once and the total number of chunks respectively. Each chunk then **undergoes a series of processing steps to improve the quality of the ECG signals that will be used as an input for the algorithm**. Each chunk is processed in sequence, and the results are saved to a .mat file named *preprocessed_ecg_data_chunk_<index>.mat.*

As mentioned in our summary of changes, we applied several checks and balances to the signals apart from the authors' mentioned, for each record in a chunk, the following checks are performed:

i. **Lead Count**: Must have exactly 12 leads.
ii. **Signal Length**: Must be 5000 samples (10 seconds at 500 Hz).
iii. **NaN Chec**k: If any value is NaN, the entire record is skipped.

A **bandpass filter was** applied in order to reduce the noise. Low frequency noise below **0.05 Hz** may represent **slow drifts generated by patient movement**, respiration or changes in electrode contact as well as from skin impedance changes. It may also represent low frequency disturbances from temperature changes or equipment. High frequency noise above **150 Hz** may be generated by **muscle artifacts, instrumental noise** or even

[3]WFDB toolbox for MATLAB and octave.

environmental noise. Therefore, **a second order butterworth filter with cutoff frequencies** 0.05 Hz and 150 Hz was applied. To apply this, the *filtfilt* function is employed to perform zero-phase filtering, ensuring no additional phase shifts.

In addition, a **high pass butterworth filter** with cutoff frequency **0.5 Hz** was added to remove low-frequency components associated with baseline wander, i.e slow oscillations that might remain after bandpass filtering.

The **arPLS method was applied for baseline correction**, the code calls a function *remove_baseline_arpls* - whose key parameters include *lambda* = 1e7 (to control how smooth the baseline should be) and *max_iter* = 10 (the maximum number of iteration updates) - to apply removal of low frequency drifts in the data which can make true features of interest difficult to analyse.

It iteratively adjusts the baseline by solving a penalized least-squares optimization problem and updating weights for robustness. As baseline drifts are removed, interpretability and feature extraction are improved (detection of QRS complex, R peaks, PR intervals, etc). Misclassification due to misleading patterns during the learning process is also avoided. In particular arPLS accounts for the asymmetry present in ECG signals and is more robust to noise avoiding unwanted distortion in the signal. In addition, it avoids overfitting the baseline to the data by penalizing large changes in the baseline estimate.

After the baseline correction, **signals are converted to millivolts** by applying a known conversion factor (4.88 microvolts per bit, then converting microvolts to millivolts). Additionally, another check is implemented; that of no sample exceeding ±10 mV. Records failing this check are skipped, as extremely large amplitudes are typically non-physiological and/or indicate artifacts/equipment errors.

As we've mentioned, at this point some of the signals continued to have a high SNR, for which we implemented **wavelet denoising** was added as an extra filter to enhance the signal quality by selectively reducing noise while preserving the key features of the ECG such as QRS complexes, R peaks, and PR intervals. The key feature of this method is that **noise that may overlap with the frequency bands of interest**, such as muscle artifacts or powerline interference[4], **is effectively targeted without distorting** the underlying morphology of the ECG signal leading to a clean signal with a significantly lower SNR and better overall signal quality. This implementation was done via a custom function *wavelet_denoise*, whose parameters are:

- Wavelet family: *sym8* - Referent to the **Daubechies** mother wavelet, selected for its ability to effectively capture the sharp transitions in ECG signals, especially around QRS complexes.

[4] Indeed, the 50 Hz electrical network noise would be let in by our initial bandpass, whose high cutoff frequency is designed to maximize our QRS complexes' clinical applicability.

- Decomposition level: 5 - Chosen to balance detail and approximation, allowing effective noise reduction without losing essential signal features.
- Denoising method: *SURE* - Utilized for its statistical approach to minimize mean squared error, providing reliable noise estimation.
- Threshold rule: *Soft* - Applied to gently reduce noise coefficients, preserving the integrity of the ECG signal's morphology.
- Noise estimate: *LevelDependent* - Allows adaptive noise estimation across different decomposition levels, enhancing the denoising effectiveness across various frequency bands.

Finally, **a min-max normalization** (as in the paper) was applied to prepare the data for the deep learning model. The raw ECG signals may have varying amplitude ranges due to patient characteristics or recording conditions which might generate erroneous results or become biased toward larger numerical values as the algorithm is applied. The min-max normalization process rescales the values of the signals into a range [0;1] **while preserving the original distribution and relationship of the data**. Therefore, **faster convergence is attained and numerical stability is achieved**, while also ensuring consistency across all remaining ECG signals. Regular standardization was not applied as it rescales to have a mean of zero and a standard deviation of one, as such it wouldn't retain the shape or distribution of data; making it less interpretable.

3.2.1. Outlier Removal

The p**rocessed signals are then classified into useful and useless signals** by the third script, which looks for files named *preprocessed_ecg_data_chunk_*.mat*. These latter useless signals are eliminated and are not used as input for the algorithm.

This is done by finding the .mat files with names matching the pattern *preprocessed_ecg_data_chunk_*.mat* and sorting them numerically by the chunk index for sequential processing. It uses the function *regexp* to extract numeric indices from filenames. Output folders are also defined as useful records are saved in the filtered_chunks folder and the useless records are saved in the dropped_records folder for further analysis. Several aspects were considered in order to classify them:

- **Baseline shifts**: the mean value is calculated and if it is outside of the defined parameters [0.03 , 0.97] then significant baseline shifts are considered to be present.

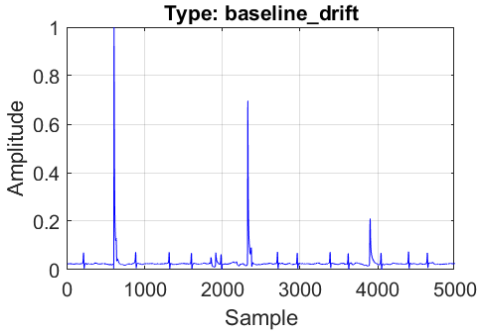

Figure 1: Graphed ECGs showing an example of baseline drift affecting the QRS peaks

- **Spike Detection**: if the difference between consecutive samples exceeds the predefined parameter 0.5 then large spikes are flagged.

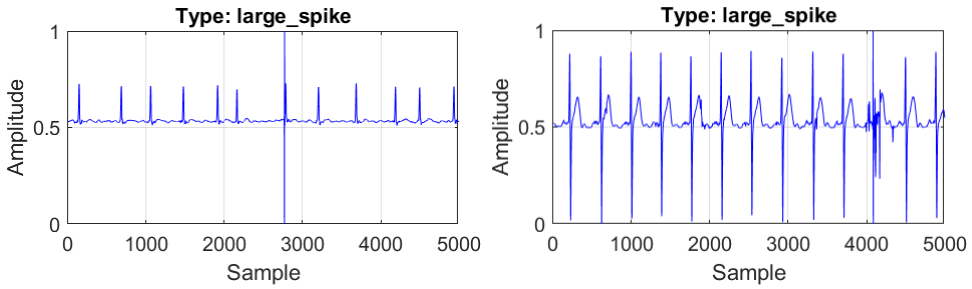

Figure 2: Graphed ECGs showing examples of large artifact spikes

- **Flatline**: signals that have low variability after normalization - which is indicated by a sliding window method that checks standard deviation in windows of size 250 samples with 125-sample overlap. If all windows have a very low standard deviation (below 0.001, the predefined parameter) standard deviation - are considered to possibly be flatlines and are classified as useless. A signal is also considered to be flatlined if its range is lower than 0.05.

- **Amplitude Range**: if the range exceeds the predefined parameter 1.1 mV, we consider it an artifact.

- **Standard Deviation**: Must be between 0.005 and 0.25 (median values of ECGs).[5]

  We set the standard deviation boundaries between 0.005 and 0.25 to remove outliers in the ECG data. This range was chosen based on the expected variability of normal ECG signals, ensuring that low and high variability signals are excluded. This approach improves data quality and aligns with common practices in ECG preprocessing workflows.

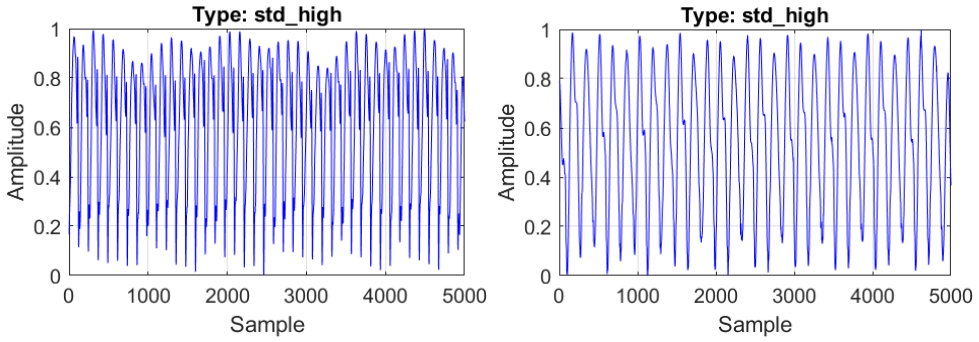

Figure 3: Graphed ECGs showing examples of invalid standard deviation values

- **Heart Rate** (HR): Must be between 40 and 200 beats per minute, estimated from R-peaks detected on lead II. This detection is done via a 5–15 Hz bandpass,

[5] The mean and standard deviation of normalized ECG signals.

derivative, squaring, moving average, and identifying peaks with a threshold-based approach; the detected R-peak indices are used to calculate beats per minute.

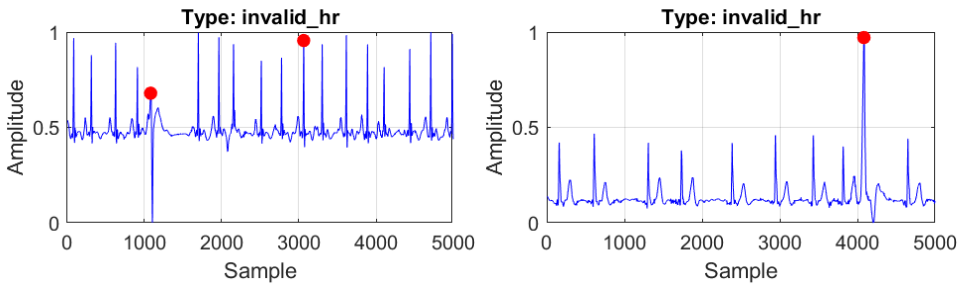

Figure 4: Graphed ECGs showing examples of invalid HR values

For each record in each chunk, the script checks every lead against the above criteria. **If any lead fails, the record is dropped.** The drop reason and which lead failed are saved in a structure of *dropped_records* under categories like baseline_drift, large_spike, flatline, amp_range, std_low, std_high, or invalid_hr. On the other hand, the good-quality signals are saved with filenames like <*chunk_filename*>*_filtered.mat* in the filtered_chunks folder.

Additionally, the last script can automatically generate plots of dropped signals at specified intervals (e.g., every 2nd, 25th, or 10th occurrence). This helps quickly inspect the nature of dropped records and verify that the criteria function as intended.

By combining these scripts, every ECG is first filtered and normalized, then verified against stricter thresholds. Signals that pass all checks remain in the final dataset; those failing one or more criteria are grouped by outlier type for potential later review.

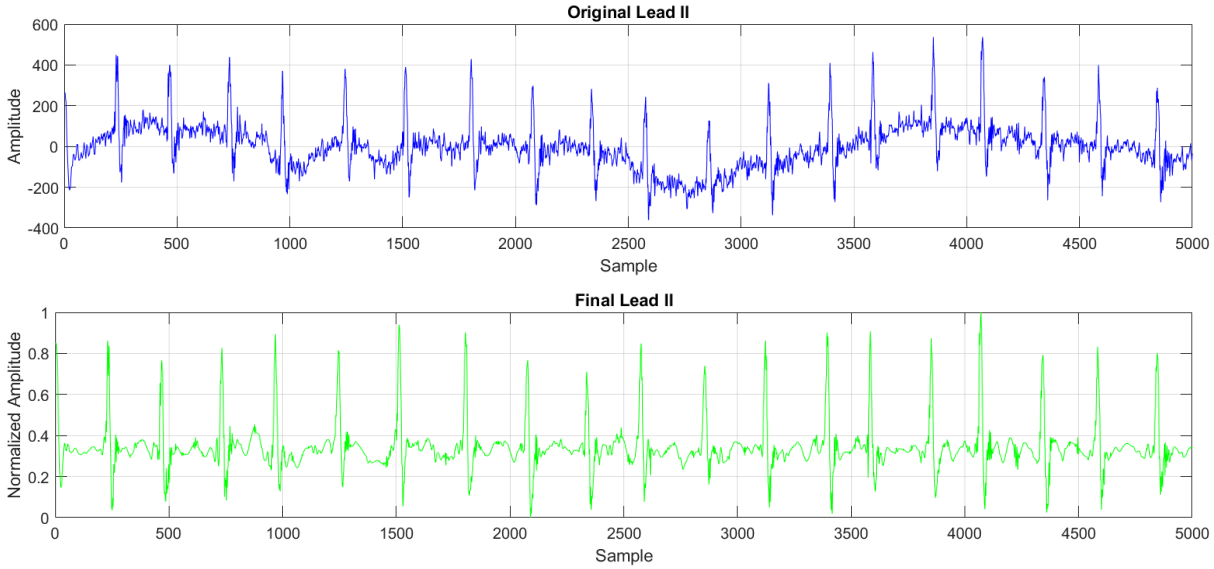

Figure 5: Sample lead before (top) and after (bottom) being passed through preprocessing pipeline.

3.3. Linear Regression to find best 3 combination

This section describes how we use linear regression (LR) to determine which three leads are most effective at reconstructing a specific "target" lead. The method is conceptually similar to the approach in the reference paper, but we introduce GPU acceleration and multiple train/test splits to obtain a more reliable RMSE estimate.

3.3.1. Overview of the paper's approach

In the referenced paper, the authors propose reconstructing a "target" ECG lead by combining three other leads via a linear regression model. They systematically test every possible combination of three leads from the remaining 11 leads, train the model with a portion of the data, and then compute the root mean square error (RMSE) on a held-out test set. Whichever combination achieves the lowest RMSE is considered best for predicting the target lead. The paper uses a single 80% training, 20% testing split and normalizes all signals into a fixed range to enhance model consistency.

3.3.2. Our GPU-accelerated method

We structure our ECG data as a three-dimensional array: number of records, number of leads, and samples per lead. **For each target lead that we wish to reconstruct, we generate all possible sets of three other leads**, excluding the target lead itself. As mentioned, rather than relying on a single train/test split, **we repeat the splitting process several times**. For each split, **around 80% of the records are used for training and 20% for testing**; this repeated splitting helps ensure that our average RMSE result is not overly influenced by one particular partition of the data split.

In each split, **we fit a standard least-squares model that estimates how to combine the three chosen leads to reconstruct the target lead**. We then predict the target lead in the test data and measure the RMSE. Afterward, we average the RMSE across all the train/test splits for that particular three-lead combination.

With this output we can see and extract the three-lead sets that produce the lowest average RMSE. Additionally, by systematically checking all three-lead subsets, we ensure no potentially better combination is overlooked. As this process can be time-consuming, we perform these calculations on a GPU for faster execution, especially useful when dealing with many records or longer signals.

3.4. Deep-learning model architecture and training details

3.4.1. Original approach

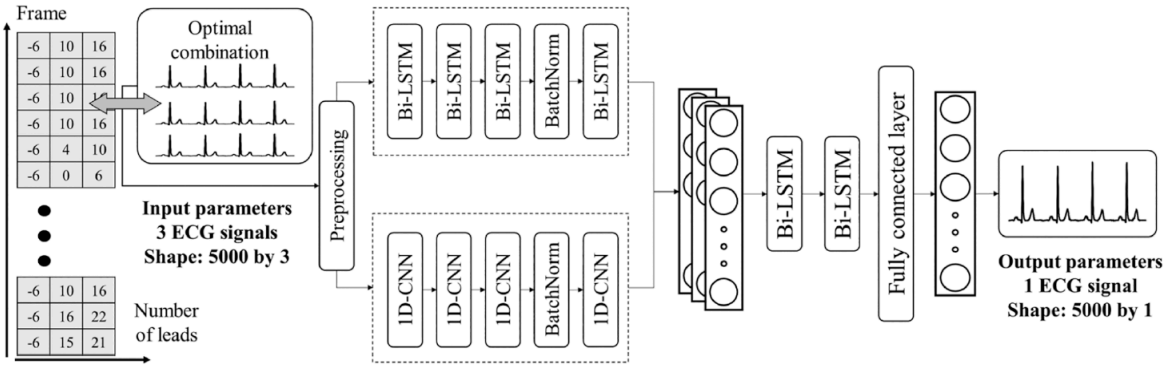

Figure 6: Schematic of the original work's modeling pipeline[6]

1. Inputs
    - The paper takes three ECG leads (each 5000 samples long) as input, for a total input shape of (5000 × 3).
2. Architecture outline:
    - The paper's diagram shows two main processing branches:
        1. A Bi‑LSTM branch: Several stacked Bi‑LSTM layers (with Batch Normalization in between) to capture temporal/sequence‑level dependencies in the ECG.
        2. A 1D‑CNN branch: Multiple 1D convolution layers (again with Batch Normalization) to learn local patterns or features in the signal.
    - These two branches' outputs are then merged (some form of concatenation or addition) and further processed by another Bi‑LSTM layer or two.
    - Finally, the network applies a fully connected layer to produce the reconstructed ECG lead (shape (5000 × 1)).

Table 1: Advantages vs disadvantages of the original work's approach.

Advantages	**Downsides**
The CNN layers **effectively capture local morphological features in the ECG** (e.g., QRS shape), while the Bi‑LSTMs help **model the long‑range temporal dependencies** of the heartbeat sequence.	Stacking multiple Bi‑LSTMs can lead to a **large number of parameters**.
By going forward and backward in the sequence, **the model can leverage both "past" and "future" context in reconstructing a missing lead.**	In practice, we found that training can take upwards of **4 hours** on the same hardware for an epoch (or for a single run) because LSTM layers can be more expensive computationally and do not parallelize as efficiently as CNN or Transformer blocks.
	The repeated Bi‑LSTM and large CNN stacks can make the **model memory‑heavy**.

[6] Yoo, H., et al.

3.4.2. Our approach: U-Net + Transformer Model

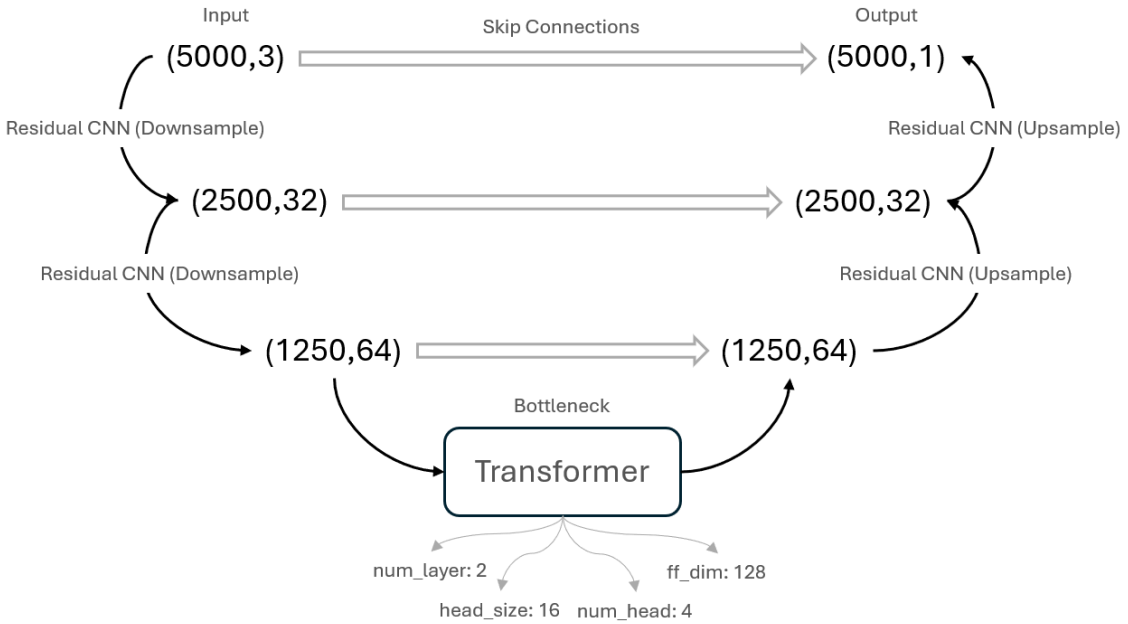

Figure 7: Simplified visualization of our model

Our alternative model also takes in 3 ECG leads of length 5000 (shape (5000 × 3)), but it **replaces the Bi-LSTM/CNN arrangement with a U-Net style encoder-decoder plus a Transformer bottleneck**.

The U-Net style Encoder/Decoder has three main components: an **Encoder** - which **applies two "residual" CNN blocks, each downsampling by a factor of 2** (after two blocks, we have downsampled from 5000 → 2500 → 1250 in the temporal dimension) - a **Decoder** - which then **upsamples step-by-step** (back from 1250 → 2500 → 5000) using transpose operations or UpSampling1D layers, with skip connections from the encoder (the hallmark of U-Net) and the **Transformer bottleneck** (located in between the encoder and decoder) - instead of more LSTMs, we feed the compressed 1D feature maps (shape around (1250, 64)) into a Transformer (multi-head self-attention). In particular, the latter works by:

1. Projecting the compressed features (64 channels) up to a certain model_dim (num_heads(4) × head_size(16)) so that the multi-head attention can operate effectively.
2. We add Positional Encoding, then stack a few Transformer encoder blocks. Each block includes:
    i. Layer Normalization
    ii. Multi-Head Attention
    iii. Residual connection
    iv. Feed-forward network (Dense + activation)
    v. Another residual connection

3. The Transformer learns global context across the entire compressed timeline (1250 steps instead of 5000) for more efficient sequence-level modeling.
4. Finally, we project back to 64 channels to match the decoder's expected dimensionality.

After this pipeline ends in the decoder reconstructing our signals back to length 5000, we add a Conv1D layer of size 1×1 to produce (5000 × 1) as the final restored ECG. This overall provides us with marked advantages for our particular dataset:

Table 2: Advantages and their respective justification our approach poses respective to the authors' for our new dataset.

Advantage	**Explanation**
Fewer Parameters	The encoder-decoder pattern with residual blocks is quite parameter-efficient, and Transformers (especially in a bottleneck) can scale well with smaller dimension settings. The original model had 1,543,969 parameters while our model has only 373,865 parameters.
Much Faster Training	In our experiments, the entire model was trained in ~20 minutes, compared to 4 hours for the paper's Bi-LSTM+CNN approach.
Better Reconstruction Accuracy	We achieved lower RMSE for the restored ECG lead than the original model on our dataset. That suggests it generalizes well and reconstructs the ECG morphology more precisely.
Multi-Scale + Global Context	The U-Net structure captures multi-scale features (thanks to downsampling + skip connections), while the Transformer bottleneck sees the global relationships across the entire heartbeat timeline.

All in all, by swapping out the Bi-LSTMs in favor of a U-Net + Transformer design, we achieve both faster training times and improved reconstruction metrics, demonstrating that sequence-to-sequence ECG tasks can be handled efficiently with modern CNN/attention architectures.

3.4.3. Training Process

As the original authors do, we base our final validation on the custom root mean squared error (RMSE) function, which squares the difference between prediction and target, averages, and takes the square root. A small epsilon is added to avoid numerical issues if the MSE is near zero.

3.4.3.1. Model Creation and Compilation

- **Architecture**: A U-Net–Transformer network is built using a specified number of Transformer layers, heads, and feed-forward dimensions. A dropout rate and L2 regularization factor are set to help prevent overfitting.
- **Optimizer**: Adam with a learning rate of $1e^{-4}$ is used. An optional SGD optimizer is shown but not the default here.
- **Metrics**: Cosine similarity (to measure waveform alignment) and built-in RMSE are monitored.
- **Compilation**: The model uses the custom RMSE loss, Adam, and the above metrics.

3.4.3.2. Training and Callbacks

- **Early Stopping**: Terminates training if validation loss fails to improve for a set number of epochs (patience). Restores best weights.
- **ReduceLROnPlateau**: Lowers the learning rate by a set factor if validation loss plateaus.
- **Checkpoint**: Saves the best model based on validation loss.
- **Fit Procedure**: Runs up to 100 epochs with a batch size of 8, returns a training history for further analysis.

4. Results and analysis

Table 3: Results of the Linear Regression to find the best 3-lead-combination for each lead

	Best Combination of Paper	RMSE (µV) of paper	Our Best Combination	Our RMSE (µV)
Lead I	aVR, aVL, V6	0.09	aVR, aVL, V5	0.175
Lead II	aVR, aVF, V6	0.065	aVR, aVF, V6	0.134
Lead III	Lead II, aVL,aVF	0.096	aVR, aVL, aVF	0.198
aVR	lead I, lead II, V6	0.091	Lead II, aVL, V1	0.315
aVL	lead I, lead III, aVR	0.108	Lead I, Lead III, aVR	0.218
aVF	lead II, lead III, aVR	0.071	Lead II, Lead III, aVR	0.144
V1	lead 1, aVR, V2	0.112	Lead I, aVR, V2	0.237
V2	V1, V3, V4	0.078	Lead III, V1, V3	0.179
V3	V2, V4, V5	0.075	Lead II, V2, V4	0.167
V4	V2, V3, V5	0.074	aVR, V3, V5	0.130
V5	V3, V4, V6	0.061	aVR, V4, V6	0.101
V6	aVR, V4, V5	0.073	Lead I, Lead II, V5	0.125

As we can see, both the ideal combinations and RMSE values vary between both approaches. The ideal combinations for each lead have certain variations but maintain relatively common tendencies (usually including at least the closest other lead). On the other

hand, our RMSE values are consistently a bit less than double the original paper's (we can see this trend in both Table 3 and 4), unfortunately, as we have no way to compare our database to theirs we can't confidently assert what could be the root cause of the problem.

Despite this, overall we have obtained satisfying reconstructions of the 12-lead signals from their 11-lead counterparts; with the RMSE values being mostly lower than 0.2. In terms of cosine similarity (see Table 4), however, we have much more satisfying values - and overall equal or better values.

As cosine similarity quantifies the preservation of the signal's fundamental characteristics during restoration, and RMSE our overall differences between the model and the original, we can extrapolate from the fact that we have a higher cosine similarity and RMSE than the original study, that our modelled ECGs overall better conserve their shape (due to the higher cosine similarity) but have more noticeable local distortions (due to the higher RMSE). This could very well be due to our use of wavelet denoising, which - with Daubechies mother waveforms - can be considered a gold standard of ECG processing due to its waveform being very similar to the QRS complex.

However, as mentioned, we have no way of reasoning the change in RMSE, and considering the extra processing measures we implemented, along with our stricter outlier rejection, it is probable that the issue is due to the intrinsic difference between the used datasets. Part of the contributing factor could be that our dataset has specific pathological labels via the SNOMED-CT vocabulary (as elaborated on in 2.2.1.), while the original dataset bases its labels on the Minnesota Classification.

Table 4: Lead-by-Lead RMSE results of the Deep Learning Models from both approaches

	Paper's Cosine Similarity	Paper's RMSE (μV)	Our Cosine Similarity	Our RMSE (μV)
Lead I	0.992	0.033	0.991	0.059
Lead II	0.993	0.0.022	0.993	0.047
Lead III	0.990	0.027	0.995	0.050
aVR	0.990	0.028	0.998	0.066
aVL	0.988	0.036	0.994	0.063
aVF	0.989	0.019	0.995	0.042
V1	0.995	0.074	0.991	0.101
V2	0.992	0.046	0.995	0.078
V3	0.989	0.045	0.994	0.074
V4	0.990	0.040	0.993	0.059
V5	0.991	0.035	0.993	0.047
V6	0.984	0.042	0.985	0.055

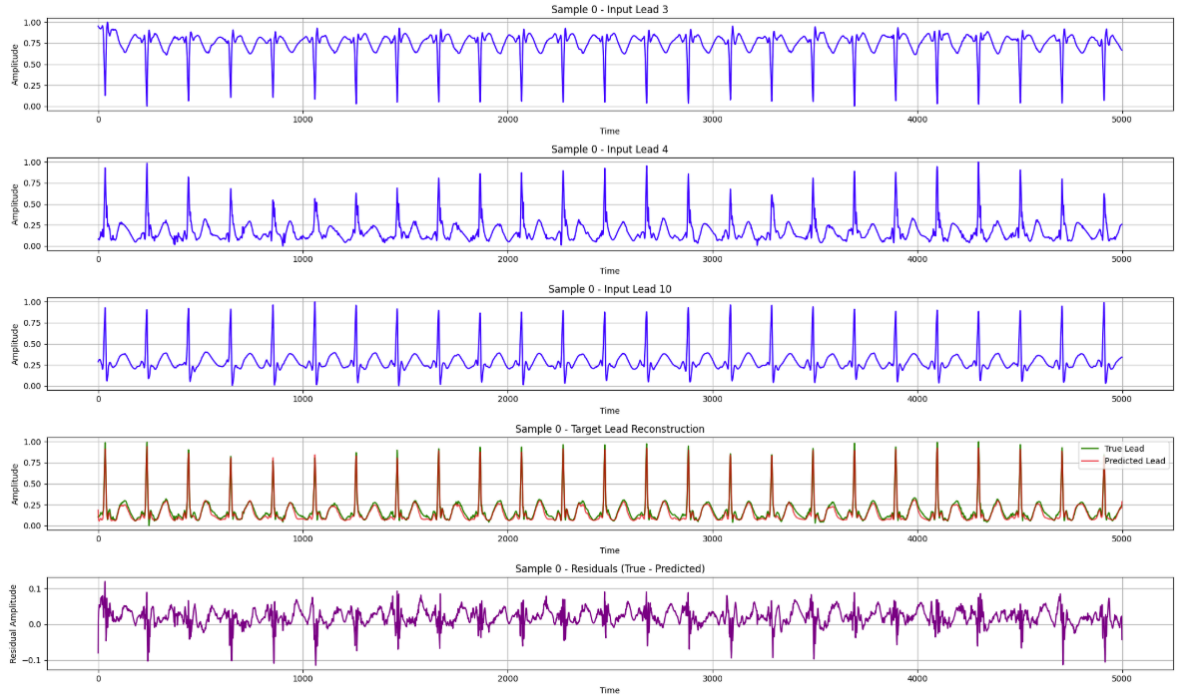

Figure 8: Example of model's performance on validation data for reconstructing "Lead I" using three leads aVR, aVL, and V5

Retrospectively, some measures we could have taken, given more time, to better extrapolate conclusions from these results are:

- Validating our algorithm against more databases.
- Validating it against the original's author database.

However, with respect to the original work we've optimized the process to be able to be run on standard laptops accessible to students, and not hospital-level technologies, while also cutting down on the processing time with comparable results.

Looking to the horizon, we could take this project further and model specific pathologies. As our dataset is labelled with SNOMED-CT, it would be possible to include or use other models or Web Services that are also connected to that vocabulary, opening up this algorithm to use by other researchers.

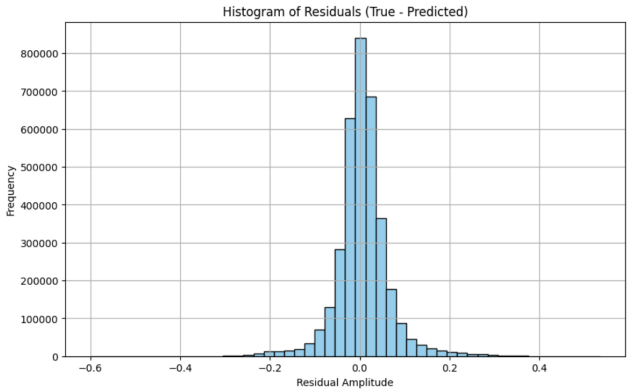

Figure 9: Histogram of our signals' residuals, as we can see, they tend to zero.

5. References

1. Yoo, H., Yum, Y., Park, S., Lee, J. M., Jang, M., Kim, Y., Kim, J.-H., Park, H.-J., Han, K. S., Park, J. H., & Joo, H. J. (2021, November 8). Kurias-ECG: A 12-lead electrocardiogram database with standardized diagnosis ontology. KURIAS-ECG: a 12-lead electrocardiogram database with standardized diagnosis ontology v1.0. https://physionet.org/content/kurias-ecg/1.0/ [Accessed the 16/12/2024]
2. Zheng, J., Guo, H., & Chu, H. (2022, August 24). *A large scale 12-lead electrocardiogram database for arrhythmia study*. A large scale 12-lead electrocardiogram database for arrhythmia study v1.0.0. https://physionet.org/content/ecg-arrhythmia/1.0.0/WFDBRecords/#files-panel [Accessed the 16/12/2024]
3. WFDB toolbox for MATLAB and octave. (n.d.). https://archive.physionet.org/physiotools/matlab/wfdb-app-matlab/ [Accessed the 17/12/2024]
4. The mean and standard deviation of normalized ECG signals. (n.d.). https://www.researchgate.net/figure/The-mean-and-standard-deviation-of-normalized-ECG-signals_fig1_45603248 [Accessed the 20/01/2025]
5. Yoo, H., Yum, Y., Kim, Y., Kim, J.-H., Park, H.-J., & Joo, H. J. (2023). Restoration of missing or low-quality 12-lead ECG signals using ensemble deep-learning model with Optimal Combination. *Biomedical Signal Processing and Control*, *83*, 104690. https://doi.org/10.1016/j.bspc.2023.104690

All Tables and Figures 1, 2, 3, 4, 5, 6, 8 and 9 are of our own making via MATLAB and Powerpoint.

6. Appendix: User Manual

Prerequisites

1. Dataset: Download the ECG dataset from PhysioNet
2. MATLAB: Ensure MATLAB is installed with all required toolboxes.
    a. The WFDB Toolbox can be downloaded from https://archive.physionet.org/physiotools/matlab/wfdb-app-matlab/ , additionally, the Signal Processing, Wavelet and Statistic and Machine Learning Toolboxes from MATLAB are needed.
3. Python: Install Python and the necessary libraries.

Step-by-Step Instructions

1. Extract the ECG Data

    1. Open the *Database_Extraction.m* file in MATLAB.
    2. Locate the *root_folder* variable in the script and set its value to the directory where the downloaded dataset is stored. This should ideally be the same directory where the .m files are located.
    3. Run *Database_Extraction.m* in the MATLAB interface.
        - This will create a .mat file named *extracted_ecg_data.mat* in the directory.

2. Preprocess the Data

    1. Open the *Preprocessing.m* file in MATLAB.
    2. Run the script.
        - This will generate 46 chunks of preprocessed data and save them in a specific directory.

3. Assess Data Quality

    1. Open the *assess_record_quality.m* file in MATLAB.
    2. Run the script.
        - This will process the previously created chunks, remove outliers, and save the cleaned data in a directory called filtered_chunks.

4. Use the Filtered Data in Python

    1. Navigate to your Python project directory.
    2. Open the provided Python notebook or script.
    3. Ensure the Python *script/notebook* is configured to load the preprocessed and filtered data from the filtered_chunks folder.
    4. Execute the Python code to begin further analysis or model training.