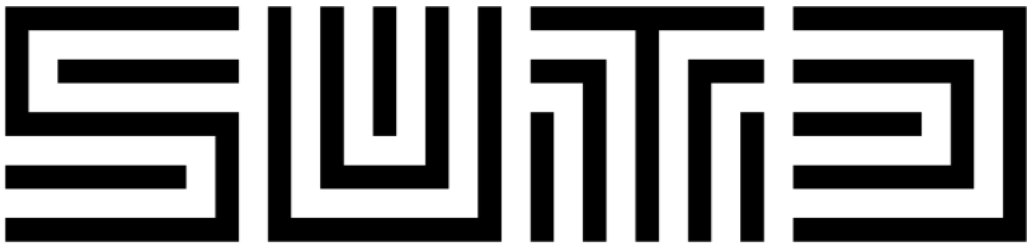


SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

50.039 Theory and Practice of Deep Learning
Group 7
Final Project Report
ECG diagnoses classification

Name	Student ID
Marcus Chao Yan	1005905
Xie Jia Cheng	1005911
Liew Min Hui	1006166

Table of Contents

Introduction

Cardiovascular diseases remain one of the leading causes of death worldwide, with early and accurate detection being a critical factor in improving patient outcomes. Electrocardiograms (ECG) are one of the most widely used diagnostic tools for detecting heart conditions, offering a non-invasive method to monitor the electrical activity of the heart. The standard ECG has 12 leads, with each lead representing the different electrical activity of the heart. However, the interpretation of ECG signals requires significant expertise and can be time-consuming, which can delay diagnosis and treatment especially in resource-limited settings. This project explores the application of deep learning techniques to identify heart conditions using 12-lead ECG data, aiming to improve diagnostic efficiency and accuracy, potentially supporting faster clinical decision-making and improved patient care.

Problem Investigated

Task

The task is to identify heart conditions using 10-second 12-lead electrocardiogram (ECG) data. It is a multi-class (multiple different types of diseases), multi-label (some patients may have multiple diseases) problem.

Dataset

The dataset was found through PhysioNet, a collection of datasets designed to support current research and stimulate new investigations in the study of complex physiologic and clinical data. Specifically, the dataset used is from the research paper A large scale 12-lead electrocardiogram database for arrhythmia study. It is a large-scale ECG recording database of 45,152 patients in Zhejiang, China. Each input is also labelled with the heart conditions of the patient, if any.

Inputs

Each input is a 10-second ECG recording from 1 patient. There are 12 channels of data per input, all recorded on the same time series. Each channel is numeric data, sampled at 500Hz.

Output

The output is a prediction of whether a particular patient has a specific heart condition. While there are a total of 63 different conditions within the original dataset, after data cleaning (elaborated on in the next section), this investigation focuses on 5 conditions – namely Sinus Bradycardia (SB), Sinus Rhythm (SR), Atrial Flutter (AF), Sinus Tachycardia (ST), T wave Change (TWC). Thus, the output will be a 1D vector with 5

values ranging from 0 to 1, representing the probability that the patient has each condition.

Data Cleaning and Pre-Processing

The dataset downloaded from PhysioNet was in the format of Waveform Database (WFDB), whereby every ECG is represented by a tuple of two files: a mat-file containing the binary raw data, and a corresponding header file with the same name and hea-extension containing the patient's condition code(s). To facilitate preprocessing and integration into a deep learning pipeline, it was necessary to convert and restructure the dataset into a more accessible and usable format. The dataset was restructured to be (1) a folder of individual CSV files, each containing 12-lead ECG signals for one patient and labelled with a unique patient ID (represented in the *data/ecg* folder), and (2) a master CSV file mapping each patient ID to their corresponding diagnosis or diagnoses *(.csvs within the data/ folder)*.

An exploratory data analysis was conducted to better understand the distribution of diagnoses within the dataset (See *scripts/EDA.ipynb* file). This included plotting the frequency of each condition, which revealed a significant class imbalance – some diagnoses appeared in many patients, while many others were rare, as shown in the plot below.

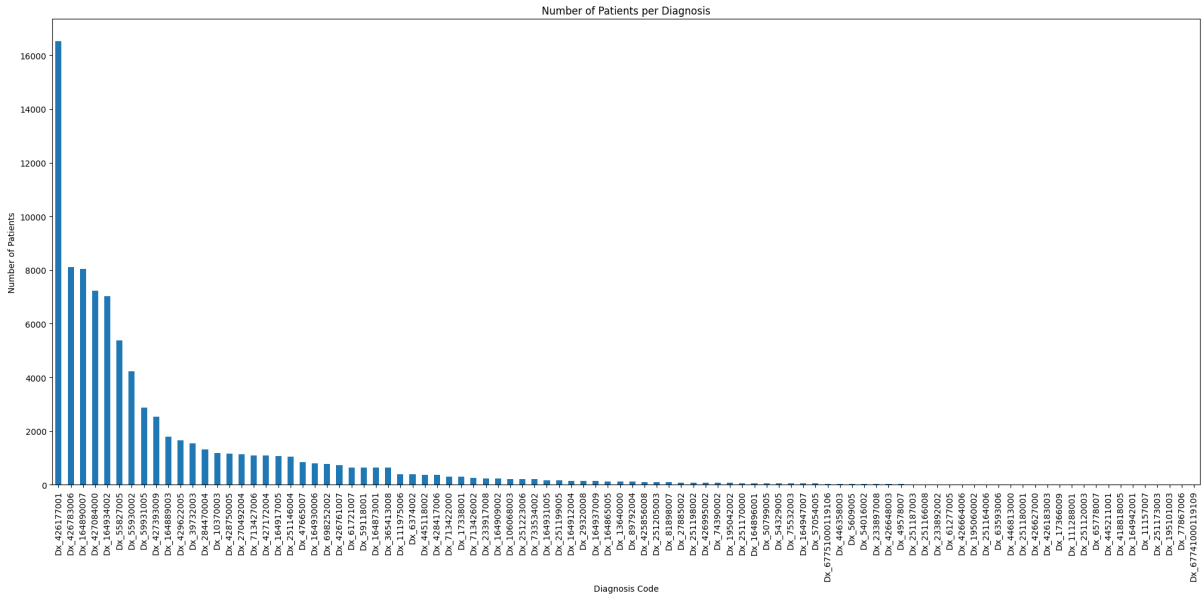


Figure 1: Frequency of Each Condition

Such imbalance poses a challenge for deep learning models, as they tend to become biased toward majority classes, resulting in poor performance on underrepresented conditions. To address this issue and ensure more reliable model training, we initially filtered the dataset to include only the diagnoses that appeared in at least 10% of patients. This reduced the number of target classes to 6, allowing the model to focus on

sufficiently represented conditions and improving its ability to generalise and make meaningful predictions.

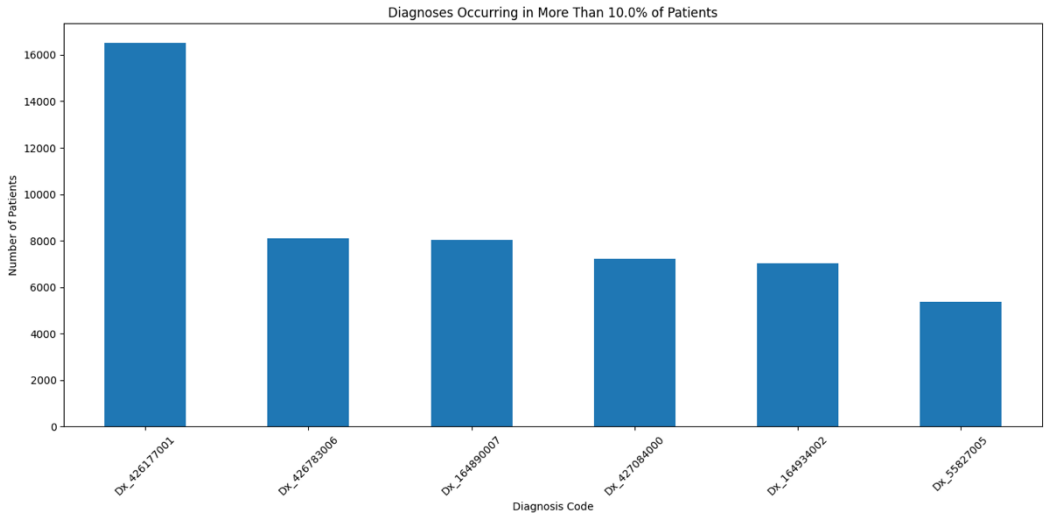


Figure 2: Distribution of 6 Target Classes

During preprocessing, some ECG CSV files were found to contain missing values (NaN) in one or more leads (See *scripts/input_checking.ipynb* file). These incomplete records were removed from the dataset, as missing data in time-series signals can compromise model performance and introduce inconsistencies during training. Corresponding entries in the master CSV file, which maps each patient ID to their diagnosis or diagnoses, were also deleted to maintain consistency between the signal data and labels. This data cleaning process was carried out in the *scripts/data_cleaning.ipynb* notebook.

To further mitigate class imbalance in the dataset, a custom iterative undersampling approach was implemented. The target number of samples for each label was set to the minimum count across all the selected diagnoses, ensuring that no diagnosis was overrepresented. For each label, patient records were randomly selected – without replacement – until the number of instances for each diagnosis reached the target. Due to the multi-label nature of the data, label frequencies were tracked across all diagnoses, ensuring fair representation across overlapping conditions. This strategy helped reduce class dominance and prevented the model from being biased toward overrepresented conditions. The iterative undersampling process was implemented in the *scripts/iterative_undersampling.ipynb* notebook.

Given the large initial dataset, reducing the records did not raise concerns, as the final dataset still contained a sufficient number of samples for each diagnosis. After cleaning and undersampling, the dataset consisted of a balanced set of approximately 7036 samples per condition, with each diagnosis representing around 24% of the total data. This ensured that the model could learn from a sufficiently diverse set of examples while addressing the class imbalance.

The resulting dataset used is thus (1) a folder of reduced individual CSV files, each containing 12-lead ECG signals for one patient and labelled with a unique patient ID, and (2) updated master CSV file mapping each patient ID to their corresponding diagnosis or diagnoses named *data/diagnoses_balanced.csv*.

The final change we made to the original dataset was to clip the lead values. As seen in Figure 3 below, there were a significant number of leads outside of the [-1,1] range, where most ECG lead values tend to reside.

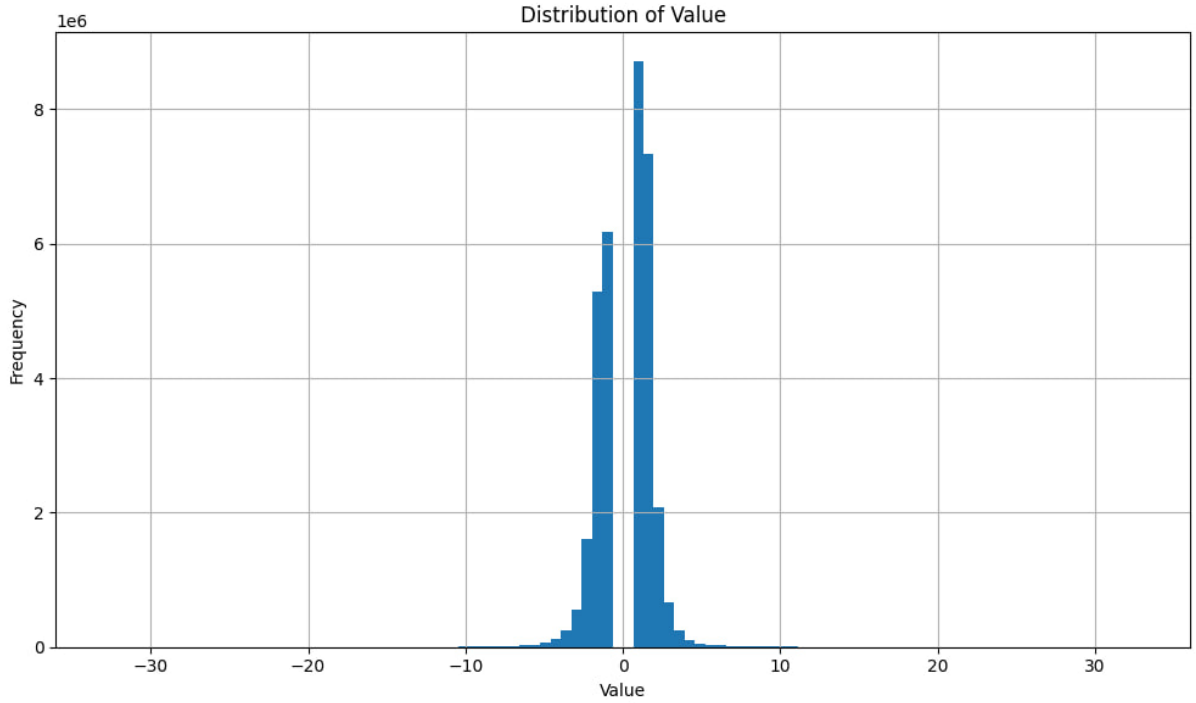


Figure 3: Distribution of Lead Values outside of range [-1,1]

To resolve this, we created a notebook *scripts/lead_value_clipping.ipynb*, which read through each lead's data across all ECG files, calculated the mean and standard deviation of each of the leads, then clipped all values which deviated ±5 standard deviations from the mean to generate a new set of ECG files, found in *data/ecg_clipped* (the cleaned data is to be downloaded following instructions on GitHub README). The mean and standard deviations of each lead, alongside their respective clipping bounds, can be found in the *metadata* folder.

Model Architecture

We proposed and trained a transformer-based ECG deep neural network for the task, with convolution-based embeddings.

Embedding Model – Convolution

The embedding model aims to do two main things:

1) Create more data channels using the initial 12 channels
2) Allow the embedding vector at any time step to contain “vicinity” information from data points forward and backward in time.

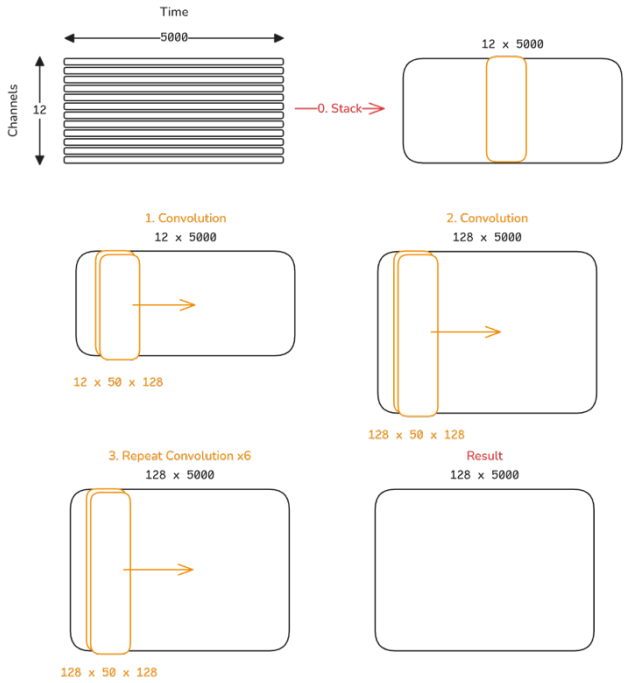


Figure 4: CNN embeddings

The initial data has 12 channels (as there are 12 leads in the ECG test), with 5000 points per channel due to a 10-second duration and a 500Hz sampling rate. We stack and treat this initial data as a 12x5000 vector. A total of 8 sets of 1-dimentional convolutions across the time axis are carried out. The first convolution set has a window size of 50, with an input dimension of 12 and output dimension (number of kernels) of 128. With same padding and 1 stride, this results in a 128x5000 vector. Functionally, this creates 128 channels from the initial 12. The second convolution set has the same window size of 50, but both input and output dimensions are 128. This retains the same vector size, but seeks to allow each vector to carry information from its vicinity.

Since each convolution is conducted along the time dimension, the resulting vector at each time point is expected to be embedded with data from earlier and later in time. With the window size of 50 in the time axis and the convolution repeated 8 sets, the resultant vector is embedded with information +- 200 time steps. With the dataset’s 1000Hz sample rate, this vicinity width is hence 400ms, roughly the amount of time for 1 heartbeat in the ECG data. The vicinity width and number of convolution repetitions can be tuned to how the model if performing.

In initial trainings of the model, this convolution embeddings model was determined to be too big. With 8 layers and 5,928,448 parameters in total, it far outweighs the main transformer at 265,605 parameters. Especially since the input data at each time step

has only 12 channels, this was possibly too big a model to train. We hence experimented with smaller versions of the same model, including shrinking the window size, output vector size and number of repetitions to no better results. The smallest we tested has 86,528 parameters.

We hence made a simpler embeddings model as a baseline to compare against. The new embeddings model still uses 1d convolutions, but with a window size of 1. This effectively does not embed vicinity data into the resultant vector, but still allows for more data channels to be created from the data's 12. Another technique inspired by CNNs is to increase the output size of the convolutions gradually. Rather than jumping from vector size 12 straight to 128, a doubling from 12 to 24 to 48 to 96 then 128 is used. The new model has 18,632 parameters, and has no noticeable difference in training from the small original embeddings model.

Transformer Model – Encoder-Only Multi-Label Classification

The main model chosen is an encoder-only transformer, using the original encoder architecture proposed by Vaswani, et al. in the 2017 paper "Attention is All You Need". A classification head, which is a simple fully connected linear layer is put on the end of the encoder layer. The output logits are put through a sigmoid layer and a threshold of 0.5 is used to set whether the prediction for that class is a positive or negative diagnosis.

Due to limitations in our training hardware, we only use 2 stacked encoders with 4 heads each. We would ideally experiment with larger encoder stacks with more heads.

Evaluation Methodology

Since the dataset is so large, we do not use epochs as checkpoints, instead checkpointing every 512 batches. To evaluate the model at that checkpoint, we different evaluation metrics to monitor the progress of training. For the multi-label classification task, the simplest method is to calculate the Hamming distance between the prediction and target as a loss. This quantifies the percentage of predictions are wrong, regardless of what class the prediction is and whether the prediction is positive or negative. This can be calculated as a total for each evaluation, giving a quick sanity check for training improvements.

We also monitor each class separately to identify if there are any problems with any particular classes. Since we consider each class separately, each is a classification problem between positive and negative. We hence use recall, precision and F1 score for each class as the primary evaluation method. Owing to the medical context, these metrics provide a reliable measure of the model's ability to correctly identify each condition, minimising the risks of missed or incorrect diagnoses. The true and false, positive and negative counts are tracked for each evaluation, and the confusion

matrices for each class are also plotted for our checking. The results are discussed below.

Training Optimizations

Several techniques were employed to help the model train.

Batch Size & Gradient Accumulation

With our limited resources, our best training machine is a 16GB GPU. With the above model parameters, the largest batch size for our data that can fit within 16GB is 4. When training with a batch size of 4, gradient updates are noisy and training is slower. Hence, we implemented gradient accumulation that accumulates losses and hence gradients across *m* batches before updating parameters. This is equivalent to training with a batch size of 4*m, while still retaining only 4 batches in memory. This technique allows us to experiment with what equivalent batch size best reduces noise in gradient updates while respecting the hardware limitation.

The losses of the batches inside each accumulation group are averaged to keep loss in the same scale as if the batch size is 4. This keeps losses in the same scale no matter the effective batch size. We settled on an accumulation size of *m*=8, for an effective batch size of 32.

Mixed Precision Training

Another technique employed to make effective use of limited training memory is mixed precision training, where lower precision data types are used for calculations that do not need higher precisions. We attempted to use PyTorch's Automatic Mixed Precision package to conduct training in FP16. However, we observed NaN gradients every ~20 batches. Based on other user reports online, we postulate that parts of model, possibly the batch norm in the transformer, are underflowing to NaNs. Furthermore, there were no notable improvements in memory usage; we were still limited to the same batch size of 4.

Adam & Learning Rate Scheduler

We employed the Adam optimizer for its robustness and adaptability to sparse gradients, which are common in multi-label classification problems. Through empirical tuning, we found that a learning rate of 5e-4 provided a good balance between convergence speed and stability. Earlier experiments with higher rates (e.g., 1e-3) led to unstable loss curves, while lower rates significantly slowed convergence. To enhance adaptability, we implemented a ReduceLROnPlateau scheduler that monitors the macro-averaged F1 score on the validation set. If performance plateaus for two checkpoints, the learning rate is halved. This mechanism allowed the model to adaptively refine learning during later training phases without manual intervention. The

full implementation of this system can be found in the Trainer class (*run.py* file), where the optimizer, scheduler, and dynamic checkpointing are configured.

Positive Weights in Loss Calculation

As mentioned in the preprocessing segment, iterative undersampling was performed such that there was an even representation of each diagnosis, with each diagnosis representing around 24% of the total data. However, this results in an imbalance between the number of positive labels and negative labels within each class.

To resolve this, we introduced **positive weights** into our BCEWithLogitsLoss function, which scales the loss contribution from positive samples in each class according to their relative scarcity. These weights are computed using the ratio of negative to positive samples for each class (neg / pos) and are passed as the pos_weight argument in the loss function. This adjustment helps the model place more emphasis on correctly identifying underrepresented diagnoses. The implementation of this weighting strategy can be found in the get_pos_weights() method within the ECGDataset class (*dataset.py* file) and is applied during training in the Trainer class (*run.ipynb* file).

Set-up Required to Run the Model

Everything required to run or modify the model is available on our GitHub repository. The dependencies are available in the *requirements.txt*. Training was done with a CUDA-enabled GPU. The cleaned dataset in the CSV format is available separately on our own hosting due to GitHub's limitations with large files. Instructions to download the data is found on our GitHub README file.

How to Retrain the Model

Before retraining the model, it is important to note that we have also included a "Settings" code block which allows you to configure model hyperparameters, data paths, and save paths.

From Scratch

Once setup, you can simply run the *run.ipynb* file from the start until the "Start from 0" markdown block. Running the code below that block with create an instance of the model and the trainer, then invoke the train method.

The trainer class is configured to save every *checkpoint_interval*, into the *model_progress* folder.

From our Saved Weights

Similar to training the model from scratch, run the *run.ipynb* file from start until the "Start from 0" markdown block. Do not run the code block below that. Instead, run the code block below the "Resume from a Checkpoint" markdown. This runs the model from the weights saved in *training_progress/clipped_balanced3/checkpoint_ep7_b4773.pt*. These is also the set of model weights we are using to evaluate the model in this report.

Should you wish to view the loss graphs, F1 scores, and confusion matrices generated by our model training, you can simply run the *run.ipynb* notebook up to the model training blocks, skip the 2 model training blocks mentioned above, then continue to run the remaining code blocks within the notebook.

Model Results

Final Model Results

The final model results are based on the checkpoint `training_progress/clipped_balanced3/checkpoint_ep7_b4773.pt`. Throughout training, we observed a consistent decrease in loss, which gradually plateaued around this checkpoint. As seen in Figure 5, the training loss exhibited general downward momentum with fluctuations, indicating that the model was learning effectively but began to plateau after approximately 35,000 batches.

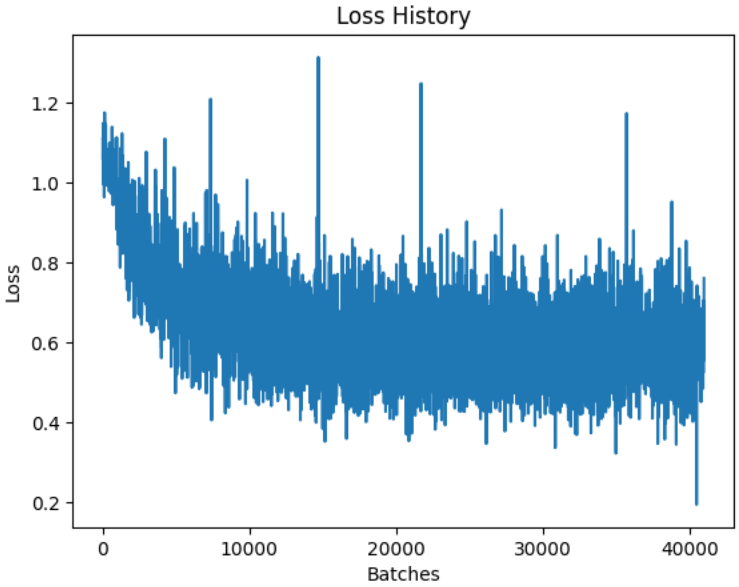


Figure 5: Loss History over training batches

Although we balanced the dataset such that each diagnosis class was equally represented (~24% of the dataset), we observed uneven performance across classes

during training. As shown in Figure 6, Class 0 (Dx_426177001) significantly outperformed the other classes in terms of F1 score, plateauing around 0.8, whereas the others hovered near 0.6.

Interestingly, Class 0 was also the most frequent class in the original (unbalanced) dataset before our balancing procedure. We therefore suspect that this could be due to label co-occurrence (correlation) being high. For instance, if Class 0 frequently co-occurs with other labels, the model may have learned more robust features for it early on.

To address this disparity and improve overall classification performance, we performed threshold tuning. Since this is a multi-label task, a single sigmoid threshold (e.g., 0.5) is not necessarily optimal across all labels. By tuning the decision threshold for each class individually – using the validation dataset F1 score as the optimization target – we were able to bring underperforming classes to comparable levels, which can be seen in the sharp improvements at the end of training.

This highlights the importance of tailored post-processing in multi-label classification problems, especially when label correlation and past class imbalance influence model calibration.

With this in mind, we identified the optimal thresholds per class to be the following: [0.75 0.7 0.55 0.75 0.5].

Figures 6 and 7 below represent the F1 score and the Hamming loss per class over the training batches. As we can see, all values do exhibit noise before converging at around 30,000 batches.

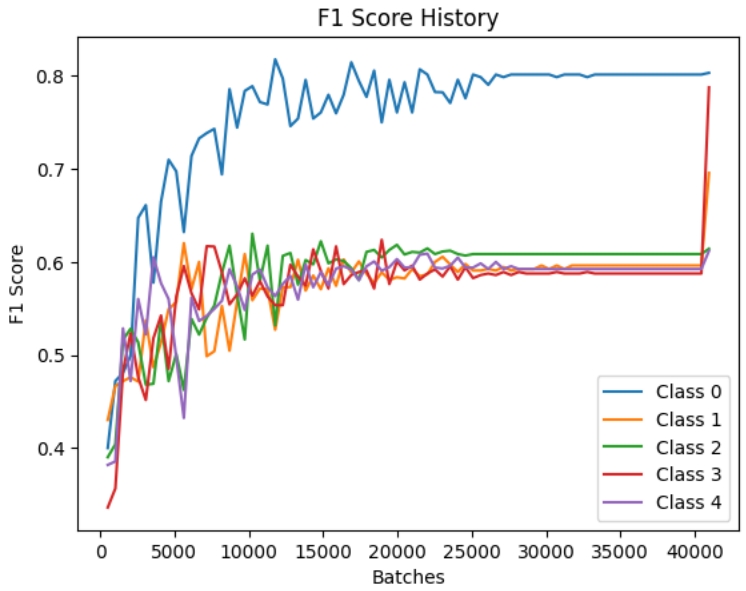


Figure 6: F1 Score History over training batches

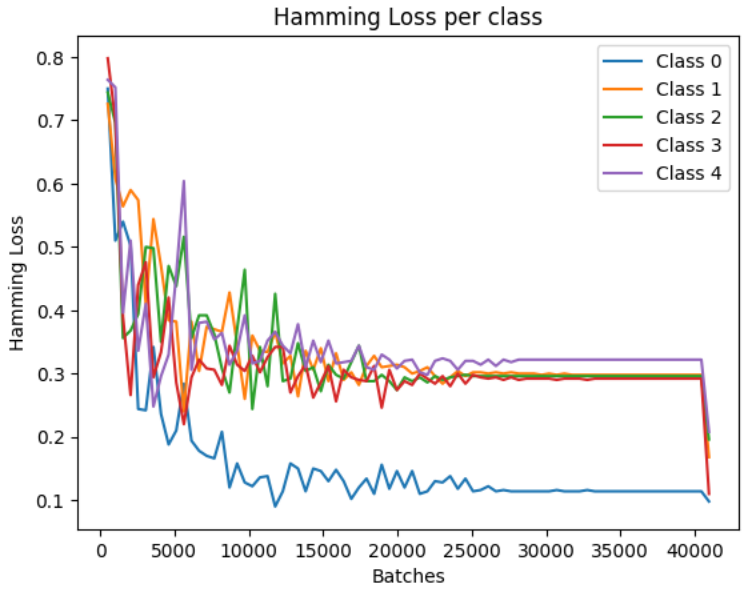
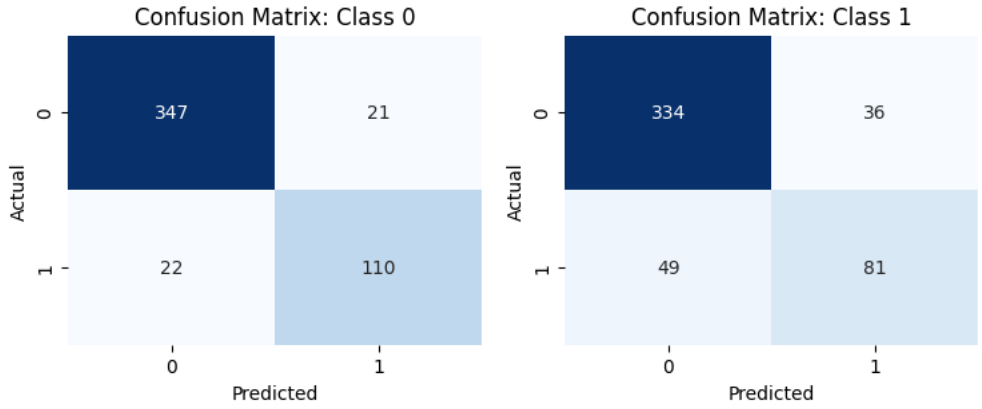
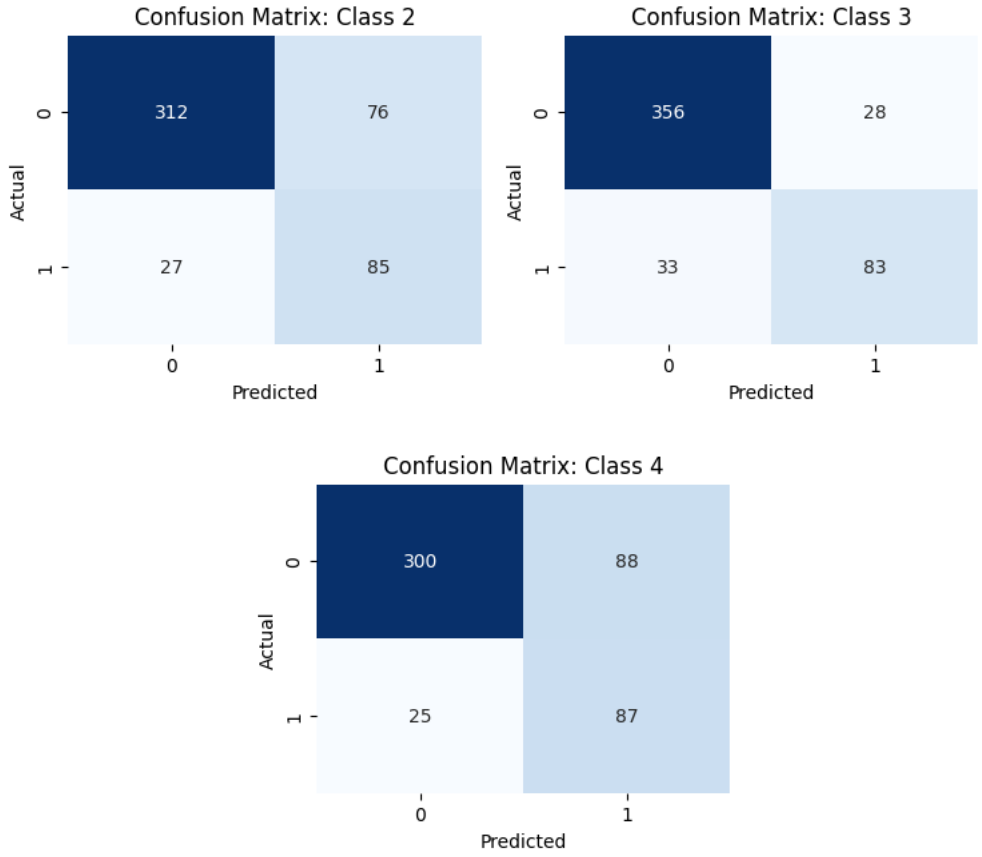


Figure 7: Hamming Loss per class over training batches

Finally, we generated confusion matrices for each of the 5 classes below. The class names correspond to the following diagnoses:

Diagnosis Code	**Given mapping in ConditionNames_SNOMED-CT.csv**	**Class Name in Confusion Matrices**
Dx_426177001	SB,Sinus Bradycardia	Class 0
Dx_426783006	SR,Sinus Rhythm	Class 1
Dx_164890007	AF,Atrial Flutter	Class 2
Dx_427084000	ST,Sinus Tachycardia	Class 3
Dx_164934002	TWC,T wave Change	Class 4





With this, we get the eventual scores:

Class Name	**Code**	**F1 Score**	**Precision**	**Recall**	**Hamming Loss**
Sinus Bradycardia (SB)	Dx_426177001	0.8032	0.8000	0.8065	0.098
Sinus Rhythm (SR)	Dx_426783006	0.6957	0.6316	0.7742	0.168
Atrial Flutter (AF)	Dx_164890007	0.6142	0.5417	0.7091	0.196
Sinus Tachycardia (ST)	Dx_427084000	0.7876	0.7907	0.7846	0.110
T wave Change (TWC)	Dx_164934002	0.6119	0.5062	0.7736	0.208
Overall	—	—	—	—	**0.156**

These results indicate that the model performs particularly well on the Sinus Bradycardia (SB) and Sinus Tachycardia (ST) classes, achieving F1 scores above 0.78 and relatively low Hamming Loss. These two classes also have strong balance between

precision and recall, suggesting confident and consistent predictions. In contrast, classes such as Atrial Flutter (AF) and T wave Change (TWC) exhibited lower precision despite reasonable recall, indicating a tendency toward over-prediction. This reinforces the importance of threshold tuning in addressing per-class calibration and improving predictive reliability across all diagnoses. Overall, given the time frame and expertise we had, we believe that the model performs well. However, we are aware that healthcare diagnosis is a high-impact task, and this model is nowhere near the expected performance of a model of this type.

Issues Identified and Future Improvements

Dataset Size

One of the biggest issues we faced was just the sheer size of the ECG dataset. Even after filtering down to five conditions, each input was a 12-channel, 5000-timestep time series, which quickly ballooned in memory usage during training. With limited GPU memory (16GB), we had to make trade-offs between batch size and model complexity. That was why we ended up using gradient accumulation and keeping batch sizes small. It worked, but training was slower and noisier than we would have liked.

Multi-label classification

We have not explored this type of problem in depth before, so it was a challenge to understand some of the issues we were facing. Unlike multi-class classification, where the model picks one label out of many, here each sample could have multiple diagnoses at once – and that made everything from loss calculation to evaluation trickier. For example, early on, we assumed a 0.5 threshold would be fine for all labels, but this led to certain classes being consistently under-predicted. It took a while to realize that each class needed its own threshold and that metrics like accuracy were not that useful. We had to dive deeper into metrics like F1 score and Hamming loss per class to understand how our model was doing. Overall, this type of task required more nuanced tuning and forced us to rethink how we interpret model output.

Embeddings model

Our difficulties with the embeddings model suggest there is more to be done in this area. The embeddings model is critical in extracting relational information from the ECG data, which could help identify some of the heart conditions. Some form of “vicinity” information needs to be embedded in each data point.

One possible solution is to create embeddings from patches of data across multiple time steps rather than data at a single time step, based on Dosovitsky et al.’s vision transformers (2020). For example, a patch can be 40ms wide. A heartbeat that is 400ms

long would be broken up into 10 patches, where a patch might contain the peak and another the trough.

Another possible improvement is to include the patient's profile in the embeddings model. The dataset includes age and gender data that could help the model.

Comparison against State-of-the-Art Models

Most current models for ECG classification focus on 1-lead ECG data, or target general or broad arrhythmia classes. Our proposed model includes several advancements. Firstly, it utilises full 12-lead ECG signals, capturing spatially distributed cardiac activity across the chest and limbs, which provides a richer and clinically relevant data representation. Secondly, it targets a focused but diverse set of arrhythmias, including underrepresented yet diagnostically important categories such as T wave Change (TWC), which are seldom modelled explicitly in existing works.

Model	Classes	ECG Leads	Architecture	Performance
MSW-Transformer	5 classes	12 lead	Transformer	- average macro-F1 scores of 77.85%, 47.57%, 66.13%, 34.60%, and 34.29%, and - average sample-F1 scores of 81.26%, 68.27%, 91.32%, 50.07%, and 63.19%
Hybrid CNN-Transformer	4 – 5 classes	1 lead	CNN + Transformer	- 97.8% accuracy on the Icentia11k dataset - 99.58% accuracy on the MIT-BIH dataset
Proposed Model	5 classes	12 lead	Transformer + Convolution-based Embedding Layer	F1 scores of 0.8032, 0.6957, 0.6142, 0.7876, 0.6119 for each respective class

Table 1: Comparison with Existing Models

Conclusion

This project set out to build a deep learning model that could help classify heart conditions using 12-lead ECG data. Along the way, we had to overcome some challenges. Namely, class imbalance, multi-label noise, working around hardware limitations. We ended up using a transformer model with a convolution-based embedding layer, which gave us decent results, especially for Sinus Bradycardia and Sinus Tachycardia.

Even though all the classes were equally represented after balancing, some conditions were still harder to detect than others. We think this is probably due to how labels tend to co-occur and how some features are just more distinguishable. Threshold tuning helped close that gap a bit, and tuning things like the learning rate and loss weights definitely made a difference.

While our model is not ready for any real clinical use, it shows that even a relatively simple transformer setup can work quite well for ECGs with the right preparation. If we had more time (and a beefier GPU), we would definitely explore better ways to model heartbeats and refine the embedding step.

References

Alexey Dosovitskiy, et al. 2020. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”

Ashish Vaswani, et al. 2017. “Attention Is All You Need”