UHASSELT
KNOWLEDGE IN ACTION

Maastricht University

# Faculteit Wetenschappen
## *School voor Informatietechnologie*

master in de informatica

***Masterthesis***

***Block Term Tensor Regression for Arrhythmia Detection and Prediction on Sinus Rhythms***

**Dries Cornelissen**

Scriptie ingediend tot het behalen van de graad van master in de informatica

**PROMOTOR :**

Prof. dr. Stijn VANSUMMEREN

**BEGELEIDER :**

dr. ir. Axel FAES

**2024**
**2025**

De transnationale Universiteit Limburg is een uniek samenwerkingsverband van twee universiteiten in twee landen: de Universiteit Hasselt en Maastricht University.

UHASSELT
KNOWLEDGE IN ACTION

**www.uhasselt.be**
Universiteit Hasselt
Campus Hasselt:
Martelarenlaan 42 | 3500 Hasselt
Campus Diepenbeek:
Agoralaan Gebouw D | 3590 Diepenbeek

Maastricht University

# Faculteit Wetenschappen
## *School voor Informatietechnologie*

master in de informatica

***Masterthesis***

***Block Term Tensor Regression for Arrhythmia Detection and Prediction on Sinus Rhythms***

**Dries Cornelissen**

Scriptie ingediend tot het behalen van de graad van master in de informatica

**PROMOTOR :**

Prof. dr. Stijn VANSUMMEREN

**BEGELEIDER :**

dr. ir. Axel FAES

# Abstract

In modern healthcare, early identification and prevention of serious health problems are crucial. Increasingly, research focuses on developing artificial intelligence models capable of detecting and predicting health risks based on a person's medical history and current condition.

Within the context of cardiovascular health, AI models play an important role in predicting risks such as atrial fibrillation, sinus tachycardia, sinus arrhythmia, and other conditions. Traditionally, neural networks are often applied to patient ECGs to predict or recognise arrhythmias.

This thesis investigates the effectiveness of block-term tensor regression (BTTR) using CP Decomposition as an alternative to neural networks. BTTR is still an emerging research technique and has not yet been widely adopted, but it shows promise. Unlike neural networks, BTTR uses multi-linear algebra (multiway decoding) and is able to capture relationships between different dimensions of the data. This is useful as ECG data is often structured as a big 3D tensor. Furthermore, it offers a more interpretable "glass-box" approach, which could help medical research in the future.

Because this technique is relatively new, its accuracy in predicting cardiovascular health risks remains uncertain. Therefore, this thesis is a case study, aiming to develop the most effective BTTR model for predicting cardiovascular risks using ECG data. The performance of BTTR is evaluated to determine its usefulness within this research domain. Specifically, we mainly focus on atrial fibrillation, but the same techniques could be used with other arrhythmias.

# Acknowledgements

In this section, I would like to thank two people. First, my co-supervisor and mentor, Dr. Ir. Axel Faes, a postdoctoral researcher in Biomedical Data Sciences at UHasselt. The topic was originally his, as he conducts research in Block-Term Tensor Regression, and he kindly offered it to me. He guided me throughout the entire process by having short meetings every two weeks. In doing so, he supported me from the initial research phase, through development, and finally during the writing of the thesis. He was always supportive and helpful, and remained positive even when classification results were disappointing.

Secondly, I would like to thank Prof. Dr. Stijn Vansummeren, a professor of Computer Science at the Data Science Institute of Hasselt University. He was my official promotor, overseeing the thesis. Although our contact was limited, as this work was more closely related to Dr. Axel Faes's research area, I appreciated his positive feedback and advice during the intermediate presentation.

# Summary

Cardiovascular diseases (CVDs) remain the leading cause of mortality worldwide, with arrhythmias such as atrial fibrillation (AF) representing a major concern due to their potential to trigger life-threatening events like stroke and heart failure. Early detection of such conditions is vital, yet traditional methods often fall short, particularly because arrhythmias tend to occur sporadically and may remain asymptomatic until they reach a dangerous stage. This unpredictable nature is especially problematic: arrhythmic episodes may not occur during routine medical check-ups, and individuals without noticeable symptoms are unlikely to seek medical attention, allowing the condition to go undiagnosed and untreated for extended periods.

While wearable ECG devices and continuous monitoring devices have already improved detection rates using Artificial Intelligence (AI), most current systems focus on identifying arrhythmias as they happen. Ideally, you want to predict these arrhythmias, allowing for timely intervention, reducing the likelihood of life-threatening outcomes. Predicting arrhythmic events in advance, especially from otherwise normal sinus ECGs, remains a significant challenge. AI, and in particular neural networks, have shown promise in detecting arrhythmias, as well as predicting future arrhythmias from current sinus rhythms. These models are excellent at learning subtle patterns in complex data but suffer from limited interpretability and high computational demands.

This thesis investigates whether an alternative approach, known as Block-Term Tensor Regression (BTTR), can address these issues. BTTR is based on tensor decomposition techniques, which allow for the modelling of multi-dimensional data such as ECGs in a structured and interpretable way. Unlike deep learning models, BTTR is closer to a 'glass-box' system, meaning it can show insights into which patterns it sees. This makes it more suitable for clinical use where interpretability and transparency are essential.

For this thesis, we will answer the following research question: **"How effective is Block-Term Tensor Regression in detecting and predicting atrial fibrillation from multi-lead ECG, and how can its pipeline be optimised for best performance?"**.
So specifically, the main goal of this research is to evaluate the effectiveness of BTTR in two tasks: (1) classifying AF from raw ECGs, and (2) predicting the risk of future AF episodes based on healthy ECG segments. While neural networks dominate current approaches, BTTR may provide a viable and interpretable alternative for ECG-based diagnostics and prediction.

The main idea behind the methodology is to break down complex ECG signals into smaller, meaningful components or patterns, and then use those as input for classification and prediction models.

To do this, we use a technique called **tensor decomposition**, which simplifies complex multi-dimensional data like ECGs. A commonly used method is **CP decomposition**, which breaks the data into a set of repeating patterns (called latent factors or components) found across the input data. The result is a set of factor matrices that describe how strongly each pattern is expressed in each dimension: samples, time and leads.

Specifically, the *sample factor matrix* shows how much each sample expresses each pattern, the *time factor* indicates which points in time are most relevant to each pattern, and the *lead*

*factor* captures which ECG leads contribute most to each pattern. This yields a compact and interpretable representation of the original signals, helping us better understand the underlying structure of the data.

A more advanced version of this technique, **Block Term Decomposition**, allows for more flexibility by capturing richer patterns by allowing components to be combined in more complex ways. We mention it for context, but keep things simple by focusing on CP decomposition.

Once these patterns are extracted, and we know which ECG samples contain which patterns, we use **classification models** to learn which patterns belong to which classes, and ultimately make predictions and classify unseen data. These models include:

- **Logistic regression**, a basic method for binary classification.
- **Random forest**, which combines many decision trees for better accuracy.
- **XGBoost**, a more powerful version that builds trees step by step and adjusts them to reduce mistakes.

As mentioned, BTTR is applied to **ECG signals**, which measure the electrical activity of the heart. Each heartbeat shows up as a set of waves: the P wave, QRS complex, and T wave. Changes in these waves can point to problems like **atrial fibrillation**, a common irregular heartbeat where normal rhythm is disrupted.

So to summarise, we break ECGs into patterns, turn those patterns into features, and use them to classify heart rhythms, all while keeping the process interpretable and manageable.

There have been numerous studies related to the classification and prediction of atrial fibrillation (AF). While our focus lies on Block Term Tensor Regression (BTTR), it's useful to understand existing approaches to AF detection and prediction in order to compare them to our method.

Early work like that of Wu et al. [1] used handcrafted features extracted from short, single-lead ECGs to detect early-stage AF. They extracted RR intervals and used Intrinsic Time-scale Decomposition (ITD) to compute entropy-based features. These features were used to train an SVM classifier. Their method performed well on short segments, reporting 95% accuracy, 96% specificity, and 93% sensitivity on Physionet data [2]. However, it relies heavily on handcrafted features and only uses single-lead data.

In contrast, deep learning methods such as those by Kachuee et al. [3] and Pyakillya et al. [4] avoid feature engineering by training convolutional neural networks directly on ECG signals. Kachuee et al. also applied transfer learning by reusing learned features for a different classification task. These models also achieved strong results of approximately 95% accuracy, but depend on large labelled datasets and often lack interpretability. Most of them work on beat-level classification, not on 10-second rhythms.

Furthermore, there have also been a few studies focusing on the prediction of AF, notably those by Attia et al. [5] and Gruwez et al. [6]. Both works explored the possibility of detecting atrial fibrillation from ECGs recorded during sinus rhythm, addressing the clinical challenge that AF often occurs unpredictably and without symptoms, making it difficult to capture using standard screening methods.

Attia et al. developed a deep learning model using convolutional neural networks (CNNs) to analyse 10-second, 12-lead ECGs from a large retrospective dataset at the Mayo Clinic. Their model achieved an AUC of 0.87 using a single ECG, increasing to 0.90 when multiple ECGs per patient were included. Gruwez et al. extended this approach using data from Belgian hospitals, training their own CNN on over 490,000 ECGs from Hospital Oost-Limburg. Their model achieved similarly strong results, with AUROC values of 0.87 on internal validation and 0.86 on an external dataset from Ziekenhuis Maas en Kempen.

Both studies demonstrate that subtle atrial abnormalities related to AF can be detected even when patients are in normal sinus rhythm, enabling earlier diagnosis and potential stroke prevention. Both share the problem of interpretability due to the use of neural networks. Despite this limitation, the results show that AI models can uncover subtle patterns in healthy ECGs, indicating risk of AF, that are not visible to clinicians.

Compared to these works, our approach also focused on AF classification and prediction using 10-second, 12-lead ECGs. However, instead of relying on deep neural networks, we employed BTTR (Block Term Tensor Regression), which offers greater interpretability and requires less data to train effectively. This choice makes the classification problem more challenging, but potentially more informative and practical for real-world clinical applications where transparency and data efficiency are critical.

This study employs an experimental approach to evaluate the performance of a simple Block Term Tensor Regression (BTTR) algorithm for classifying ECG signals and predicting arrhythmias. The work begins with a diagnostic classification task, which is simpler, to optimise preprocessing, tensor decomposition, and model selection, establishing a robust pipeline. This pipeline is then adapted for prognostic prediction, which predicts future atrial fibrillation (AF) from ECGs that appear normal at recording. The primary modification for the prognostic task lies in data sampling, with the remainder of the methodology largely unchanged.

The methodology includes selecting relevant ECG data, preprocessing signals to enhance quality, and applying CANDECOMP/PARAFAC (CP) tensor decomposition to extract meaningful patterns in an unsupervised manner. These latent patterns serve as features for supervised machine learning models. Multiple classification algorithms are tested and their hyperparameters optimised via grid search. Model performance is evaluated using accuracy metrics to interpret results.

BTTR operates in two main stages: first, unsupervised tensor decomposition identifies shared latent patterns across ECG recordings. Second, a supervised classifier learns to distinguish heart rhythms based on these patterns. but firstly, we need data.

We use the PTB-XL dataset from Physionet [2, 7, 8], which contains over 21,000 annotated 12-lead ECG recordings. For this study, only recordings labelled as normal sinus rhythm (SR) or atrial fibrillation (AF) are included. Other metadata such as patient age or sex is excluded to focus solely on ECG-derived features.

Again, two related tasks are addressed: diagnostic classification (differentiating AF from SR) and prognostic prediction (predicting future AF from normal ECGs). Prognostic prediction is more challenging, as the ECGs do not contain obvious arrhythmia signs, only subtle indicators. Sampling strategies differ accordingly: classification uses all SR and AF recordings, while prediction uses SR recordings from patients who develop AF within one month alongside SR recordings from patients without AF episodes in the dataset, as proposed by Attia et al [5]. However for this task, we do have to take into account that there will be some label noise. Some AF events might be unrecorded, resulting in sinus rhythms of that patient being wrongfully classified as healthy. Overall, both tasks mostly only differ in sampling in labelling, meaning our methodology is mostly the same for both tasks.

Before we start decomposition, our raw clinical ECG signals require preprocessing to mitigate issues such as baseline drift, amplitude variability, high-frequency noise, and high sampling rates, which can distort tensor decompositions or bias them towards high-amplitude signals. Various preprocessing pipelines were evaluated, focusing on centring, amplitude scaling, and noise reduction. Effectiveness was evaluated via visualisations, reconstruction errors, and correlations of latent factors with rhythm labels.

For normalisation, global min-max scaling was initially tested, which scales the entire tensor to [0,1] based on global extrema. However, this failed to equalise effective signal ranges, allowing large-amplitude ECGs to dominate decomposition. Per-signal min-max normalization, scaling each signal independently to [0,1], mitigated this issue and improved decomposition quality by

balancing sample contributions. Per-sample normalisation across leads was also evaluated but found ineffective.

Denoising steps were crucial and included:

- A high-pass Butterworth filter (cutoff 0.5 Hz) to remove baseline wander without phase distortion.
- A 50 Hz notch filter to suppress power line interference.
- Wavelet denoising to reduce high-frequency noise while preserving essential waveform features.

These steps consistently enhanced decomposition outcomes and were incorporated into all pipelines with careful parameter tuning to avoid loss of subtle predictive information.

To reduce computational burden and redundancy from the original 500 Hz sampling (5000 points per 10-second recording), decimation was applied. This consists of a low-pass filtering followed by downsampling every $n$-th point, while avoiding aliasing. For classification, decimation by a factor of 20 preserved essential patterns, for prognostic prediction, we had to be more careful in order not to lose subtle features.

Other techniques, such as standardisation and per-signal max-absolute normalisation, were explored but did not improve decomposition quality, often introducing undesirable variability or ineffective scaling.

The final preprocessing pipeline thus consists of denoising, per-signal min-max normalisation, and decimation. Alternative tested methods either degraded performance or offered no benefits.

For decomposition, we initially considered Block Term Decomposition (BTD), due to its ability to factorise tensors into higher-rank blocks capturing complex relationships. However, there is not yet a widely accepted optimisation approach, and no implementation in Python. Since we aimed to assess the effectiveness of BTTR on our use case, rather than research BTD optimisation, we chose CANDECOMP/PARAFAC (CP) decomposition, a simpler, well-established technique that decomposes tensors into sums of rank-1 components. CP is computationally efficient, well-supported in Python (Tensorly library), and allows reproducible analyses. Both techniques aim to decompose an input tensor into patterns and latent factors, meaning our same developed methodology could also be applied to BTD.

CP decomposition was performed using Tensorly's `decomposition.parafac` on downsampled tensors approximately sized $(1000(samples), 250(Hertz{*}seconds), 12(leads))$. The rank parameter, controlling the number of latent factors, was varied between 4 and 12, with ranks below 4 causing underfitting and above 12 exceeding memory limits, with improvements plateauing around rank 8. The algorithm was run with up to 20 iterations (`n_iter_max`) and a tolerance of $1 \times 10^{-5}$ for convergence, initialised using the 'svd' method for stability and speed. Factor normalisation was enabled for interpretability, orthogonalisation was disabled to avoid restricting the solution space, and a small L2 regularisation ($l2\_reg = 10^{-3}$) improved class separation, increasing performance.

In order to evaluate our decomposition, we used correlation metrics to assess how well the extracted latent factors relate to the rhythm labels. Specifically, we computed Pearson correlation coefficients, which measure the strength of linear relationships between the latent components and the target classes. This allowed us to quantify how effectively the decomposition separated AF from sinus rhythms. Pearson correlation served as our primary evaluation metric due to its simplicity, interpretability, and direct indication of how strongly each component aligns with the classification task. Now that we know which components are the best features to differentiate the classes, component selection can be performed. Components with correlations above 0.1 were considered useful. Strong correlations exceeded 0.2.

Furthermore, we visualised the factor matrices to show how each pattern is influenced by the samples, time points, and ECG leads. This helps us see when and where important features appear in the data, making the method more interpretable than deep neural networks, which do not easily reveal what they have learned.

CP decomposition results in factor matrices for samples, time, and leads. For classification, only the sample factor matrix (size $samples \times R$) was used, representing each ECG's association to the latent patterns. Time and lead factors supported interpretability but were excluded from classifiers.

Latent factor values, typically varying between $\pm 10$, were scaled to [0,1] using `MinMaxScaler` from `scikit-learn` to avoid model bias from feature magnitude differences.

Polynomial feature expansion and Principal Component Analysis (PCA) were tested to capture nonlinear feature interactions and reduce dimensionality/noise, respectively. However, these techniques did not result in significant performance improvements and were therefore excluded from the final pipeline for simplicity and efficiency.

The sample matrix was split into training and testing sets (80/20 ratio), stratified by labels. The training set was always kept balanced in order not to prioritise the majority class while predicting. For testing we used 2 approaches: a balanced distribution and a more natural real-life distribution.

We evaluated Logistic Regression, Random Forest, and XGBoost classifiers, selecting models based on performance and tuning hyperparameters through grid search. The model performance was evaluated via cross-validation. Cross-validation metrics include accuracy, precision, recall, confusion matrix, and F1-score averaged across folds. However, since cross-validation assumes consistent data distributions, we couldn't use it as an evaluation on a naturally distributed test set. In this case, we trained and tested the model using a single train-test split.

To summarise our experiments, we first investigated the impact of different preprocessing settings on the classification task. We tested various combinations of denoising thresholds, wavelet decomposition levels, and decimation factors, while keeping all other parameters fixed. We found that the choice of preprocessing techniques in general had a much larger impact than the specific parameter values. Our final setup, using wavelet denoising at level 3 with a threshold of 7 and a decimation factor of 20, consistently led to better correlation results.

We then moved on to study the effect of the decomposition rank and $\ell_2$ regularisation. Here, we observed that stronger regularisation generally improved correlation with the rhythm labels, while higher ranks tended to reduce it. However, when we evaluated these settings in an actual classification task, the results proved otherwise. We found that higher ranks combined with moderate regularisation yielded the best performance, particularly when using Logistic Regression as the classifier. This suggests that while strong regularisation may enhance correlation, it does not necessarily lead to better classification.

Finally, we applied the same process to the prediction task, with a specific focus on optimising the preprocessing stage, as this is where we expected differences compared to classification. Since the patterns related to future AF in SR ECGs are very subtle, we tested lower decimation factors and found that reducing the amount of downsampling improved results.

For the classification task, our best model achieved an average accuracy of around 65%, with balanced overall performance. However, a more detailed analysis showed the model performed better on SR samples (precision 88%, recall 62%) than on AF samples (precision 32%, recall 69%). This may be due to greater variability in the AF class, while SR signals appear more uniform.

In the more challenging prediction task, where the goal was to predict future AF from normal-looking SR signals, we obtained an accuracy of approximately 59%. Despite the modest score, this is still above chance level, suggesting that subtle indicators of AF may indeed be present

in early SR signals. These results highlight the potential of our BTTR approach, even in data-scarce settings.

A number of key findings emerged from our study:

- Increasing the $\ell_2$ regularisation parameter improved latent factor correlations but did not improve classification, suggesting it captured common but less discriminative patterns.
- The decomposition step was the most crucial: once meaningful latent factors were extracted, even simple classifiers such as logistic regression performed strongly
- Contrary to correlation values, higher decomposition ranks improved classification performance by introducing additional information.
- Preprocessing techniques, especially amplitude scaling, denoising, and decimation, had a major impact and were crucial to get good results.

Compared to related work, our classification results fall short of reported accuracies up to 95% using handcrafted features or deep learning on the same PTB-XL dataset. Similarly, prediction tasks on large private datasets reported accuracies around 79%, while we reached 59%. Nevertheless, our method provides interpretability and requires far less data, showing promise as a lightweight and explainable alternative.

This thesis explored how tensor decomposition can be used to extract meaningful and interpretable features from ECG signals for both classification and prediction of AF. Although we did not achieve state-of-the-art performance, we demonstrated that BTTR can reveal latent structure in ECG data that supports clinically relevant tasks, even with limited data.

A key lesson was the importance of preprocessing. Early experiments suffered from poor decomposition due to neglecting signal scaling, denoising, and proper evaluation metrics. Initially, we focused on minimising reconstruction error, which did not align with downstream classification goals. Switching to correlation-based evaluation revealed more relevant features.

Another important insight was the central role of decomposition. Rather than focusing on optimising classifiers, we found that once a strong latent structure was available, even simple models sufficed. This suggests that good features matter more than complex modelling, especially in small-data settings.

From a technical standpoint, this project deepened our understanding of tensor methods applied to waveform data, particularly how preprocessing and parameter tuning impact decomposition. On the research side, it highlighted the importance of selecting appropriate evaluation metrics and carefully testing each stage of the pipeline. Interestingly, some of our most valuable insights came not from high accuracy but from understanding what didn't work and why.

As for our final thoughts, we started this thesis with the research question: **"How effective is Block-Term Tensor Regression in detecting and predicting atrial fibrillation from multi-lead ECG, and how can its pipeline be optimised for best performance?"**

Looking back, we can conclude that for now, Block-Term Tensor Regression is not as effective as deep neural networks for this task. However, the method shows potential. We have shown that it is possible to apply this approach to both the detection and prediction of atrial fibrillation from ECG data, and that it can produce useful results.

One of the main advantages we experienced is that the model offers some level of interpretability through the decomposition visualisations. While we cannot yet draw concrete conclusions from these visualisations, they give an idea of what might be possible in the future. Another benefit is that BTTR seemed to perform reasonably well even with a limited amount of data. For example, in the prediction task we only had 42 positive samples, but the model still reached an accuracy of 59%.

In summary, while BTTR does not currently outperform deep learning models in terms of raw accuracy, it offers useful properties such as interpretability and better performance with small datasets, while still giving results. This makes it an interesting approach for future research.

This future research could explore Block Term Decomposition, which may capture more complex latent patterns than CP decomposition. Applying this approach to larger, more diverse datasets could improve both classification and prediction performance. Additionally, incorporating heartbeat-level features, such as RR intervals, into the tensor representation may provide further improvements, especially in detecting early signs of AF.

# Samenvatting

Hart- en vaatziekten (HVZ) blijven wereldwijd de belangrijkste doodsoorzaak, waarbij aritmieën zoals atrial fibrillation (AF) een grote zorg zijn vanwege hun potentieel om levensbedreigende gebeurtenissen zoals beroertes en hartfalen te veroorzaken. Vroege detectie van dergelijke aandoeningen is van vitaal belang, maar traditionele methoden schieten vaak tekort, vooral omdat aritmieën de neiging hebben sporadisch op te treden en asymptomatisch kunnen blijven totdat ze een gevaarlijk stadium bereiken. Deze onvoorspelbare aard is bijzonder problematisch: aritmische episodes treden mogelijk niet op tijdens routine medische controles, en personen zonder merkbare symptomen zullen waarschijnlijk geen medische hulp zoeken, waardoor de aandoening ongediagnosticeerd en onbehandeld blijft gedurende lange perioden.

Hoewel draagbare ECG-apparaten en continue monitoringapparaten de detectiepercentages al hebben verbeterd met behulp van Kunstmatige Intelligentie (AI), richten de meeste huidige systemen zich op het identificeren van aritmieën op het moment dat ze zich voordoen. Idealiter wilt u deze aritmieën voorspellen, waardoor tijdige interventie mogelijk is, en de kans op levensbedreigende uitkomsten wordt verminderd. Het vooraf voorspellen van aritmische gebeurtenissen, vooral vanuit verder normale sinus-ECG's, blijft een aanzienlijke uitdaging. AI, en in het bijzonder neurale netwerken, hebben veelbelovende resultaten getoond in het detecteren van aritmieën, evenals het voorspellen van toekomstige aritmieën vanuit de huidige sinusritmes. Deze modellen zijn uitstekend in het leren van subtiele patronen in complexe gegevens, maar lijden onder beperkte interpreteerbaarheid en hoge computationele eisen.

Deze thesis onderzoekt of een alternatieve benadering, bekend als Block-Term Tensor Regressie (BTTR), deze problemen kan aanpakken. BTTR is gebaseerd op tensor decompositie technieken, die het mogelijk maken om multidimensionale gegevens zoals ECG's op een gestructureerde en interpreteerbare manier te modelleren. In tegenstelling tot deep learning modellen is BTTR dichter bij een 'glass-box' systeem, wat betekent dat het inzichten kan tonen in welke patronen het ziet. Dit maakt het geschikter voor klinisch gebruik waar interpreteerbaarheid en transparantie essentieel zijn.

Voor deze thesis zullen we de volgende onderzoeksvraag behandelen: **"Hoe effectief is Block-Term Tensor Regressie in het detecteren en voorspellen van atrial fibrillation vanuit multi-lead ECG's, en hoe kan de pipeline worden geoptimaliseerd voor de beste prestaties?"**
Het hoofddoel van dit onderzoek is dus om de effectiviteit van BTTR te evalueren in twee taken: (1) het classificeren van AF vanuit ruwe ECG's, en (2) het voorspellen van het risico op toekomstige AF-episodes op basis van gezonde ECG-segmenten. Hoewel neurale netwerken de huidige benaderingen domineren, kan BTTR een levensvatbaar en interpreteerbaar alternatief bieden voor ECG-gebaseerde diagnostiek en voorspelling.

De kern van de methodologie is het opsplitsen van complexe ECG-signalen in kleinere, betekenisvolle componenten of patronen, en deze vervolgens te gebruiken als invoer voor classificatie- en voorspellingsmodellen.

Hiervoor gebruiken we een techniek genaamd **tensor decompositie**, die complexe multidimensionale gegevens zoals ECG's vereenvoudigt. Een veelgebruikte methode is **CP decompositie**,

die de gegevens opsplitst in een set van herhalende patronen (de zogeheten latente factoren of componenten) die over de invoergegevens heen worden gevonden. Het resultaat is een set factormatrices die beschrijven hoe sterk elk patroon wordt uitgedrukt in elke dimensie: samples, tijd en leads.

Specifiek toont de *sample factormatrix* hoeveel elk sample elk patroon uitdrukt, de *tijdsfactor* geeft aan welke tijdspunten het meest relevant zijn voor elk patroon, en de *leadfactor* vangt op welke ECG-leads het meest bijdragen aan elk patroon. Dit levert een compacte en interpreteerbare representatie van de originele signalen op, wat ons helpt de onderliggende structuur van de gegevens beter te begrijpen.

Een meer geavanceerde versie van deze techniek, **Block Term Decompositie**, maakt meer flexibiliteit mogelijk door rijkere patronen vast te leggen, doordat componenten op complexere manieren kunnen worden gecombineerd. We vermelden het voor context, maar houden de zaken eenvoudig door ons te richten op CP decompositie.

Zodra deze patronen zijn geëxtraheerd en we weten welke ECG-samples welke patronen bevatten, gebruiken we **classificatiemodellen** om te leren welke patronen tot welke klassen behoren, en uiteindelijk om voorspellingen te doen en ongeziene gegevens te classificeren. Deze modellen omvatten:

- **Logistic regression**, een basis methode voor binaire classificatie.
- **Random forest**, die veel beslisbomen combineert voor betere nauwkeurigheid.
- **XGBoost**, een krachtigere versie die bomen stap voor stap bouwt en aanpast om fouten te verminderen.

Zoals vermeld, wordt BTTR toegepast op **ECG-signalen**, die de elektrische activiteit van het hart meten. Elke hartslag verschijnt als een reeks golven: de P-golf, QRS-complex en T-golf. Veranderingen in deze golven kunnen wijzen op problemen zoals **atrial fibrillation**, een veelvoorkomende onregelmatige hartslag waarbij het normale ritme wordt verstoord.

Dus samengevat, we splitsen ECG's in patronen, zetten die patronen om in kenmerken, en gebruiken ze om hartritmes te classificeren, dit alles terwijl het proces interpreteerbaar en beheersbaar blijft.

Er zijn talrijke studies uitgevoerd met betrekking tot de classificatie en voorspelling van AF. Hoewel onze focus ligt op Block Term Tensor Regression (BTTR), is het nuttig om bestaande benaderingen voor AF-detectie en AF-voorspelling te begrijpen om ze te kunnen vergelijken met onze methode.

Eerder werk, zoals dat van Wu et al. [1], gebruikte handmatig gemaakte features, geëxtraheerd uit korte, single lead ECG's, om beginnende AF te detecteren. Ze extraheerden RR-intervallen en gebruikten Intrinsic Time-scale Decomposition (ITD) om op entropie gebaseerde kenmerken te berekenen. Deze kenmerken werden gebruikt om een SVM-classificeerder te trainen. Hun methode presteerde goed op korte segmenten, met een gerapporteerde nauwkeurigheid van 95%, specificiteit van 96% en gevoeligheid van 93% op Physionet-data [2]. Het is echter sterk afhankelijk van handmatig gemaakte kenmerken en gebruikt alleen éénkanaals gegevens.

Daarentegen vermijden deep learning-methoden, zoals die van Kachuee et al. [3] en Pyakillya et al. [4], feature engineering door convolutionele neurale netwerken direct op ECG-signalen te trainen. Kachuee et al. pasten ook transfer learning toe door geleerde kenmerken te hergebruiken van een andere classificatietaak. Deze modellen behaalden ook sterke resultaten van ongeveer 95% nauwkeurigheid, maar zijn afhankelijk van grote gelabelde datasets en missen vaak interpreteerbaarheid. Verder classificeren ze op hartslag-niveau, niet op ritmes van 10 seconden.

Bovendien zijn er ook enkele studies geweest die zich richtten op de voorspelling van AF, met name die van Attia et al. [5] en Gruwez et al. [6]. Beide werken onderzochten de mogelijkheid om AF te detecteren aan de hand van ECG's die tijdens sinusritme waren opgenomen, en

pakten de klinische uitdaging aan dat AF vaak onvoorspelbaar en zonder symptomen optreedt, waardoor het moeilijk te detecteren is met standaard screeningsmethoden.

Attia et al. ontwikkelden een deep learning-model met behulp van convolutionele neurale netwerken (CNN's) om 10-seconden, 12-lead ECG's te analyseren uit een grote privé dataset vanuit de Mayo Clinic. Hun model behaalde een AUC van 0,87 met één enkele lead, wat toenam tot 0,90 wanneer meerdere leads per patiënt werden opgenomen. Gruwez et al. breidden deze aanpak uit met gegevens uit Belgische ziekenhuizen, waarbij ze hun eigen CNN trainden op meer dan 490.000 ECG's van Ziekenhuis Oost-Limburg. Hun model behaalde vergelijkbaar sterke resultaten, met AUC-waarden van 0,87 bij interne validatie en 0,86 op een externe dataset van Ziekenhuis Maas en Kempen.

Beide studies tonen aan dat subtiele atriale afwijkingen gerelateerd aan AF kunnen worden gedetecteerd, zelfs wanneer patiënten in normaal sinusritme zijn, wat een eerdere diagnose en potentiële beroertepreventie mogelijk maakt. Beide delen het probleem van interpreteerbaarheid door het gebruik van neurale netwerken. Ondanks deze beperking tonen de resultaten aan dat AI-modellen subtiele patronen in gezonde ECG's kunnen ontdekken die wijzen op het risico op AF, die niet zichtbaar zijn voor clinici.

Vergeleken met deze werken richtte onze aanpak zich ook op AF-classificatie en AF-voorspelling met behulp van 10-seconden, 12-kanaals ECG's. Maar in plaats van te vertrouwen op diepe neurale netwerken, gebruikten we BTTR (Block Term Tensor Regression), wat een grotere interpreteerbaarheid biedt en minder gegevens vereist om effectief te trainen. Deze keuze maakt het classificatieprobleem uitdagender, maar potentieel informatiever en praktischer voor klinische toepassingen in de echte wereld, waar transparantie en data-efficiëntie cruciaal zijn.

Deze studie maakt gebruik van een experimentele aanpak om de prestaties te evalueren van een eenvoudig Block Term Tensor Regressie (BTTR) algoritme voor het classificeren en voorspellen van aritmieën zoals AF. Ons werk begint met een diagnostische classificatietaak, die eenvoudiger is, om preprocessing, tensor decompositie en modelselectie te optimaliseren en zo een robuuste pipeline te creëren. Deze pipeline wordt vervolgens aangepast voor prognostische voorspelling, die toekomstige atrial fibrillation voorspelt op basis van ECG's die er tijdens de opname normaal uitzien. De belangrijkste wijziging voor de prognostische taak ligt in de datamonstering, waarbij de rest van de methodologie grotendeels ongewijzigd blijft.

De methodologie omvat het selecteren van relevante ECG-gegevens, het voorbereiden van signalen om de kwaliteit te verbeteren, en het toepassen van CANDECOMP/PARAFAC (CP) tensor decompositie om betekenisvolle patronen te extraheren. Deze latente patronen dienen als kenmerken voor supervised machine learning-modellen. Meerdere classificatiealgoritmes worden getest en hun hyperparameters geoptimaliseerd via een grid search. De modelprestaties worden geëvalueerd met behulp van nauwkeurigheidsmetrieken om de resultaten te interpreteren.

BTTR werkt in twee hoofdfasen: eerst identificeert de unsupervised tensor decompositie terugkomende latente patronen in ECG-opnames. Ten tweede leert een supervised classificatiemodel hartritmes te onderscheiden op basis van deze patronen, maar eerst hebben we data nodig.

We gebruiken de PTB-XL dataset van Physionet [2,7,8], die meer dan 21.000 geannoteerde 12-lead ECG-opnames bevat. Voor deze studie zijn alleen opnames gelabeld als normaal sinusritme (SR) of atrial fibrillation (AF) opgenomen. Andere metadata zoals patiëntleeftijd of geslacht zijn uitgesloten om ons uitsluitend te richten op ECG-afgeleide kenmerken.

Zoals eerder benoemd, worden er 2 gerelateerde taken aangepakt: diagnostische classificatie (differentiëren van AF en SR) en prognostische voorspelling (voorspellen van toekomstige AF op basis van normale ECG's). Prognostische voorspelling is uitdagender, omdat de ECG's geen duidelijke aritmie-tekenen bevatten, alleen subtiele indicatoren. Beide taken verschillen in hun sampling strategie: classificatie gebruikt alle SR- en AF-opnames, terwijl de predictie SR-opnames gebruikt van patiënten die binnen één maand AF zullen ontwikkelen, naast SR-opnames van patiënten zonder AF-episodes in de dataset, zoals voorgesteld door Attia et al [5].

Voor deze taak moeten we echter rekening houden met label noise. Sommige AF voorkomens kunnen ongeregistreerd blijven, wat resulteert in sinusritmes van die patiënt die ten onrechte als gezond worden geclassificeerd. Over het algemeen verschillen beide taken voornamelijk in sampling en labelling, wat betekent dat onze methodologie voor beide taken grotendeels hetzelfde is.

Voordat we beginnen met decompositie, vereisen onze ruwe klinische ECG-signalen preprocessing om problemen zoals baseline wander, amplitudevariabiliteit, hoge frequentie ruis en sampling rate te verminderen, die tensor decomposities kunnen vertekenen of beïnvloeden ten gunste van signalen met hoge amplitude. Verschillende preprocessing-pipelines werden geëvalueerd, met de nadruk op centreren, amplitudeschaling en ruisonderdrukking. De effectiviteit werd geëvalueerd via visualisaties, reconstructie errors en correlaties van latente factoren met ritmelabels.

Voor normalisatie werd aanvankelijk globale min-max schaling getest, die de gehele tensor schaalt naar [0,1] op basis van globale extrema. Dit slaagde er echter niet in om effectieve signaalranges gelijk te maken, waardoor ECG's met grote amplitude de decompositie domineerden. Per-signaal min-max normalisatie, waarbij elk signaal onafhankelijk naar [0,1] werd geschaald, verminderde dit probleem en verbeterde de decompositiekwaliteit door de bijdragen van samples in evenwicht te brengen. Per-sample normalisatie over leads werd ook geëvalueerd, maar bleek ineffectief.

Ruisonderdrukking was cruciaal en omvatte:

- Een high-pass Butterworth-filter (cut-off frequency 0,5 Hz) om baseline wander te verwijderen zonder fasevervorming.
- Een 50 Hz notch-filter om powerline interference te onderdrukken.
- Wavelet-denoising om hoge frequentie ruis te verminderen met behoud van essentiële waveforms.

Deze stappen verbeterden consequent de decompositieresultaten en werden opgenomen in alle pipelines met zorgvuldige parametertuning om verlies van subtiele voorspellende informatie te voorkomen.

Om de rekenlast en redundantie van de oorspronkelijke 500 Hz sampling (5000 punten per 10-seconden opname) te verminderen, werd decimation toegepast. Dit bestaat uit een low-passfilter, gevolgd door downsampling van elk $n$-de punt, terwijl aliasing wordt vermeden. Voor classificatie behield decimation met een factor 20 essentiële patronen, voor de predictie moesten we voorzichtiger zijn om geen subtiele kenmerken te verliezen.

Andere technieken, zoals standaardisatie en per-signaal max-absolute normalisatie, werden onderzocht, maar leidden niet tot significante prestatieverbeteringen en werden daarom uit de uiteindelijke pipeline verwijderd voor eenvoud en efficiëntie.

De definitieve preprocessing-pipeline bestaat dus uit denoising, per-signaal min-max normalisatie en decimering. Alternatieve geteste methoden verminderde de prestaties of boden geen voordelen.

Voor de decompositie overwogen we aanvankelijk Block Term Decomposition (BTD), vanwege het vermogen om tensors te factoriseren in hogere-rangs blokken die complexe relaties vastleggen. Er is echter nog geen algemeen geaccepteerde optimalisatiebenadering, en geen implementatie in Python. Aangezien we de effectiviteit van BTTR op onze use case wilden beoordelen, in plaats van onderzoek te doen naar BTD-optimalisatie, kozen we voor CANDECOMP/PARAFAC (CP) decompositie, een eenvoudigere, goed ingeburgerde techniek die tensors decomposeert in sommen van rang-1 componenten. CP is computationeel efficiënt, goed ondersteund in Python (Tensorly-bibliotheek), en maakt reproduceerbare analyses mogelijk. Beide technieken hebben als doel een invoertensor te decomponeren in patronen en

latente factoren, wat betekent dat onze ontwikkelde methodologie ook op BTD kan worden toegepast.

CP-decompositie werd uitgevoerd met Tensorly's `decomposition.parafac` op gedownsamplede tensors met een geschatte grootte van $(1000(samples), 250(Hertz * seconden), 12(leads))$. De rankparameter, die het aantal latente factoren bepaalt, varieerde tussen 4 en 12, waarbij rangen onder de 4 leidden tot onderfitting en boven de 12 de geheugenlimieten overschreden, waarbij verbeteringen afvlakten rond rang 8. Het algoritme werd uitgevoerd met maximaal 20 iteraties (`n_iter_max`) en een tolerantie van $1 \times 10^{-5}$ voor convergentie, geïnitialiseerd met de 'svd'-methode voor stabiliteit en snelheid. Factor-normalisatie werd ingeschakeld voor interpreteerbaarheid, orthogonalisatie werd uitgeschakeld om beperking van de solution space te voorkomen. Verder paste we een kleine L2-regularisatie ($l2\_reg = 10^{-3}$) toe, wat de klassenscheiding verbeterde, wat op zijn beurt dan weer de prestaties verhoogde.

Om onze decompositie te evalueren, gebruikten we correlatiemetrieken om te beoordelen hoe goed de geëxtraheerde latente factoren verband houden met de ritmelabels. Specifiek berekenden we Pearson correlatiecoëfficiënten, die de sterkte van lineaire relaties tussen de latente componenten en de doelklassen meten. Dit stelde ons in staat om te kwantificeren hoe effectief de decompositie AF scheidde van sinusritmes. Pearson correlatie diende als onze primaire evaluatiemetriek vanwege de eenvoud en directe indicatie van hoe sterk elk component de labels scheidt. Nu we weten welke componenten de beste kenmerken zijn om de klassen te onderscheiden, kan component selectie worden uitgevoerd. Componenten met correlaties boven 0,1 werden als nuttig beschouwd. Sterke correlaties overschreden 0,2.

Bovendien visualiseerden we de factormatrices om te laten zien hoe elk patroon wordt beïnvloed door de samples, tijdspunten en ECG-leads. Dit helpt ons te zien wanneer en waar belangrijke kenmerken in de gegevens verschijnen, waardoor de methode interpreteerbaarder is dan diepe neurale netwerken, die niet direct onthullen wat ze hebben geleerd.

CP-decompositie resulteert in factormatrices voor samples, tijd en leads. Voor classificatie werd alleen de sample-factormatrix (grootte $samples \times Rank$) gebruikt, die de associatie van elke ECG met de latente patronen vertegenwoordigt. Tijd- en leadfactoren droegen bij tot de interpreteerbaarheid, maar zijn niet nuttig voor de classificatie.

Deze latente waarden, die typisch variëren tussen $\pm 10$, werden geschaald naar [0,1] met behulp van `MinMaxScaler` van `scikit-learn` om feature bias door verschillen in magnitude te voorkomen.

Polynomial feature expansion en Principal Component analysis (PCA) werden getest om niet-lineaire interacties vast te leggen en respectievelijk dimensionaliteit/ruis te verminderen. Deze technieken resulteerden echter niet in significante prestatieverbeteringen en werden daarom uitgesloten van de uiteindelijke pipeline voor eenvoud en efficiëntie.

De sample-matrix werd gesplitst in een trainings- en testset (80/20 verhouding), stratified op labels. De trainingsset werd altijd gebalanceerd gehouden om de meerderheidsklasse niet te bevoordelen tijdens het voorspellen. Voor de testen gebruikten we 2 benaderingen: een gebalanceerde verdeling en een meer natuurlijke, realistische verdeling.

We evalueerden Logistic regression, Random Forest en XGBoost-classifiers, waarbij we modellen selecteerden op basis van prestaties en hyperparameters afstemden via grid search. De modelprestaties werden geëvalueerd via cross-validation (CV). Cross-validation metrieken omvatten accuracy, precision, recall, confusion matrices en F1-scores, uitgemiddeld over de CV folds. Aangezien CV echter een consistente data distributie veronderstelt, konden we het niet gebruiken als evaluatie op een natuurlijk verdeelde testset. In dit geval trainden en testten we het model met behulp van een enkele train-test splitsing.

In onze experimenten hebben we eerst de impact van verschillende preprocessing parameters op de classificatietaak onderzocht. We testten verschillende combinaties van ruisonderdrukking-thresholds, wavelet-decompositie-levels en decimation-factoren, terwijl alle andere parameters

constant werden gehouden. Uiteindelijk werd het duidelijk dat de keuze van preprocessing-stechnieken over het algemeen een veel grotere impact had dan de specifieke parameterwaarden hiervan. Onze uiteindelijke opzet, met behulp van wavelet-ruisonderdrukking op niveau 3 met een drempel van 7 en een decimation-factor van 20, leidde consequent tot betere correlatieresultaten.

Vervolgens hebben we het effect van de decompositierang en $\ell_2$-regularisatie bestudeerd. Hier zagen we dat sterkere regularisatie over het algemeen de correlatie van latente factoren met de ritmelabels verbeterde, terwijl hogere rangen deze neigden te verminderen. Toen we deze instellingen echter in een daadwerkelijke classificatietaak evalueerden, bleken de resultaten anders. We vonden dat hogere rangen in combinatie met matige regularisatie de beste prestaties opleverden, met name bij gebruik van Logistic Regression als classifier. Dit suggereert dat hoewel sterke regularisatie de correlatie kan verbeteren, dit niet noodzakelijkerwijs leidt tot betere classificatie.

Ten slotte hebben we hetzelfde proces toegepast op de predictie taak, met een specifieke focus op het optimaliseren van de preprocessing, aangezien we hier verschillen verwachtten ten opzichte van classificatie. Aangezien de patronen gerelateerd aan toekomstige AF in SR ECG's erg subtiel zijn, testten we lagere decimation-factoren en constateerden we dat het verminderen van de decimation de resultaten verbeterde.

Voor de classificatietaak behaalde ons beste model een gemiddelde nauwkeurigheid van ongeveer 65%, met een gebalanceerde algehele performance. Een meer gedetailleerde analyse toonde echter aan dat het model beter presteerde op SR-samples (precision 88%, recall 62%) dan op AF-samples (precision 32%, recall 69%). Dit kan te wijten zijn aan een grotere variabiliteit in de AF-klasse, terwijl SR-signalen uniformer lijken.

In de meer uitdagende predictie-taak, waarbij het doel was om toekomstige AF te voorspellen uit normaal ogende SR-signalen, behaalden we een nauwkeurigheid van ongeveer 59%. Ondanks de bescheiden score, ligt dit nog steeds boven het toevalsniveau, wat bevestigt dat subtiele indicatoren van AF inderdaad aanwezig kunnen zijn in SR-signalen. Deze resultaten benadrukken het potentieel van onze BTTR-aanpak, zelfs wanneer data schaars is.

Er kwamen een aantal belangrijke bevindingen uit onze studie naar voren:

- Het verhogen van de $\ell_2$-regularisatieparameter verbeterde de correlaties van latente factoren, maar verbeterde de classificatie niet, wat suggereert dat het sterke, maar minder onderscheidende patronen vastlegde.
- De decompositiestap was het meest cruciaal: zodra betekenisvolle latente factoren waren geëxtraheerd, presteerden zelfs eenvoudige classifiers zoals Logistic regression zeer goed.
- In tegenstelling tot correlatiewaarden, verbeterden hogere decompositierangen de classificatieprestaties door aanvullende informatie te introduceren.
- preprocessingstechnieken, met name amplitude-scaling, ruisonderdrukking en decimation, hadden een grote impact en waren cruciaal voor goede resultaten.

In vergelijking met gerelateerd onderzoek, waarin met handmatig geconstrueerde kenmerken of deep learning op dezelfde PTB-XL-dataset nauwkeurigheden tot 95% werden gerapporteerd, blijven onze classificatieresultaten achter. Op dezelfde manier rapporteerden voorspellingstaken op grote private datasets nauwkeurigheden van ongeveer 79%, terwijl wij slechts 59% bereikten. Niettemin biedt onze methode interpreteerbaarheid en vereist het veel minder gegevens, wat veelbelovend is als lichtgewicht en interpreteerbaar alternatief.

Deze thesis onderzocht hoe tensor decompositie kan worden gebruikt om betekenisvolle en interpreteerbare kenmerken te extraheren uit ECG-signalen voor zowel classificatie als voorspelling van AF. Hoewel we geen state-of-the-art prestaties behaalden, toonden we aan dat BTTR latente structuren in ECG-gegevens kan onthullen die klinisch relevante taken ondersteunen, zelfs met beperkte gegevens.

Een belangrijke les was het belang van preprocessing. Eerdere experimenten leden onder slechte decompositie door het negeren van signaalschaling, ruisonderdrukking en geschikte evaluatiemetrieken. Aanvankelijk richtten we ons op het minimaliseren van reconstructie-error, wat niet resulteerden in een goede classificatie. Overstappen op correlatiegebaseerde evaluatie onthulde wel relevantere kenmerken.

Een ander belangrijk inzicht was de centrale rol van decompositie. In plaats van ons te richten op het optimaliseren van classifiers, vonden we dat zodra een sterke latente structuur beschikbaar was, zelfs eenvoudige modellen volstonden. Dit suggereert dat goede kenmerken belangrijker zijn dan complexe modellering, vooral in omgevingen met weinig gegevens.

Vanuit technisch oogpunt heeft dit project ons begrip verdiept van decompositie methoden op ECG signalen, met name hoe preprocessing en parameter tuning de decompositie beïnvloeden. Aan de onderzoekszijde benadrukte het het belang van het selecteren van geschikte evaluatiemetrieken en het zorgvuldig testen van elke fase van de pipeline. Interessant is dat sommige van onze meest waardevolle inzichten niet voortkwamen uit hoge nauwkeurigheid, maar uit het begrijpen van wat niet werkte en waarom.

Uiteindelijk zijn we deze thesis begonnen met de onderzoeksvraag: **"Hoe effectief is Block-Term Tensor Regressie in het detecteren en voorspellen van atrial fibrillation vanuit multi-lead ECG's, en hoe kan de pipeline worden geoptimaliseerd voor de beste prestaties?"**

Terugkijkend kunnen we concluderen dat Block-Term Tensor Regressie vooralsnog niet zo effectief is als diepe neurale netwerken voor deze taak. De methode toont echter potentieel. We hebben aangetoond dat het mogelijk is om deze benadering toe te passen op zowel de detectie als de voorspelling van atrial fibrillation uit ECG-gegevens, en dat het nuttige resultaten kan opleveren.

Verder hebben we ook de voordelen van deze techniek ervaren. Bijvoorbeeld dat het model in zekere mate een interpreteerbaarheid biedt door middel van de decompositievisualisaties. Hoewel we nog geen concrete conclusies uit deze visualisaties kunnen trekken, geven ze wel een idee van wat in de toekomst mogelijk zou kunnen zijn. Een ander voordeel is dat BTTR redelijk goed leek te presteren, zelfs met een beperkte hoeveelheid gegevens. Zo hadden we in de predictie taak slechts 42 positieve samples, maar bereikte het model toch een nauwkeurigheid van 59%.

We kunnen dus concluderen dat, hoewel BTTR momenteel niet beter presteert dan deep learning-modellen wat betreft pure nauwkeurigheid, het nuttige eigenschappen biedt zoals interpreteerbaarheid en relatief goede prestaties met kleine datasets. Dit maakt het een interessante benadering voor toekomstig onderzoek.

Dit toekomstige onderzoek zou in plaats van CP decompositie, Block Term Decomposition kunnen verkennen, die complexere latente patronen kan vastleggen dan CP decompositie. Verder zou het toepassen van deze benadering op grotere, meer diverse datasets de classificatie- als de predictie prestaties kunnen verbeteren. Bovendien kan het opnemen van kenmerken op hartslagniveau, zoals RR-intervallen in de tensorrepresentatie, verdere verbeteringen opleveren, vooral bij het detecteren van vroege tekenen van AF.

# Contents

# Chapter 1

# Introduction

## 1.1 Background & Context

Cardiovascular diseases (CVDs) are the leading cause of mortality worldwide, responsible for millions of deaths annually. In 2019 alone, an estimated 17.9 million people lost their lives to CVDs, accounting for 32% of all global deaths. [9]
Among these, atrial fibrillation (AF) stands out as a prevalent and serious cardiac arrhythmia. AF causes the heart's upper chambers (atria) to beat chaotically and rapidly, significantly increasing the risk of life-threatening cardiovascular events such as stroke and heart failure, thereby contributing substantially to the overall CVD mortality burden. [10]
Many of these conditions, including heart attacks, arrhythmias, and heart failure, develop gradually and may remain asymptomatic until they suddenly reach an advanced stage, at which point treatment options can be limited and the risk of serious complications or death is significantly higher. Early detection is therefore crucial to allow for timely intervention, reducing the likelihood of life-threatening outcomes. [11]

Firstly, we need arrhythmia detection, but there are some challenges. Certain risk factors, such as high blood pressure and cholesterol levels, can indicate an increased likelihood of developing arrhythmias, but they do not provide a direct diagnosis. The only definitive way to detect an arrhythmia is through ECG recordings that capture an irregular heart rhythm in real-time. However, this presents a major challenge: arrhythmias are often intermittent and unpredictable, meaning they may not occur during a routine check-up.

One approach to address this issue is Holter monitoring, where a patient wears a portable ECG device that records heart activity over an extended period, usually between 24 hours and a week. While this increases the chances of detecting an arrhythmia, it remains impractical and expensive. Modern wearable devices, such as smartwatches, provide continuous ECG monitoring, but their capabilities are still limited to classification. They can detect arrhythmias only when they occur, but cannot predict them in advance.

This is a critical limitation: if an arrhythmic event is not captured, it remains undiagnosed. Since many arrhythmias occur sporadically and without symptoms, relying solely on ECGs that capture these events leaves a large diagnostic gap. This is where Artificial Intelligence (AI) plays a role.

AI has already significantly improved ECG analysis, enabling faster, more accurate, and automated detection of heart conditions such as atrial fibrillation, sinus arrhythmia, and other cardiac abnormalities. Traditional ECG interpretation relies on expert analysis, which can be time-consuming and prone to errors. AI models, particularly machine learning and deep learning techniques, can efficiently analyse large amounts of ECG data, identifying complex patterns that may be difficult for humans to detect.

AI-driven ECG analysis offers several advantages. Unlike doctors, who are prone to human error and not always available, AI provides consistent, real-time analysis. This technology is already integrated into hospitals, Holter monitors, and smartwatches, improving accessibility and efficiency.

However, the current use of AI in ECG analysis is primarily focused on detecting existing arrhythmias rather than predicting future events. Most AI models classify heart rhythms only when an arrhythmic episode is present, meaning they do not address the core issue: the lack of available arrhythmic ECGs for training and diagnosis.

To overcome this challenge, a different approach is needed—one that analyses normal (sinus) ECGs to predict the likelihood of an upcoming arrhythmic event. By identifying subtle patterns in otherwise healthy ECGs, AI could detect early warning signs that may not be obvious to human experts. This would enable early intervention, bridging the gap between reactive detection and proactive prevention.

Among AI approaches, neural networks, especially deep learning models such as convolutional neural networks (CNNs), have become the dominant method for ECG analysis. Their ability to automatically learn features from raw ECG signals makes them highly effective. CNNs excel at recognizing spatial patterns in ECG waveforms, making them well-suited for ECG classification.

## 1.2 Problem statement

Neural networks dominate ECG analysis because they outperform traditional methods in terms of accuracy, robustness, and scalability. They can detect subtle arrhythmias, classify ECG signals into different categories, and even predict future cardiac events based on normal ECGs. However, they come with significant challenges. One major drawback is their black-box nature—neural networks process ECG inputs and produce predictions or classifications without providing insight into why a particular decision was made. This lack of interpretability makes it difficult for doctors and researchers to trust or understand the model's reasoning. Additionally, neural networks require substantial computational resources, making them less practical in certain real-world applications.

These limitations have led to ongoing research into more interpretable alternatives, such as Block-Term Tensor Regression (BTTR). BTTR is a relatively new technique based on tensor decompositions and regression, offering several advantages. First, it provides more transparency into how it processes data, making it closer to a glass-box model rather than an opaque black-box. Second, it is particularly well-suited for capturing complex relationships between multiple dimensions of the data, which could be beneficial for ECG analysis.

## 1.3 Research Objectives & Approach

In this thesis, we answer the research question: **"How effective is Block-Term Tensor Regression in detecting and predicting atrial fibrillation from multi-lead ECG, and how can its pipeline be optimised for best performance?"**
We explore the effectiveness of BTTR, in predicting atrial fibrillation (AF) using normal, healthy ECGs. By focusing on predictive analysis rather than mere classification, we aim to determine whether BTTR can provide an interpretable and efficient alternative to neural networks for the early detection of arrhythmias. We assess whether BTTR can achieve comparable accuracy to existing studies that use neural networks for the same task. Additionally, we evaluate its performance using sensitivity, specificity, and the confusion matrix for a comprehensive comparison. A side note is that we weren't always able to conduct this on the same datasets, which may influence the comparison. Despite this limitation, the findings contribute valuable insights into the potential of BTTR for ECG-based arrhythmia prediction.

## 1.4 Thesis Structure

First, in Chapter 2 (*Background*), we provide essential background information. This is necessary to understand the key concepts and algorithms that were not developed as part of this work but are foundational to it. In particular, we introduce the decomposition technique and the regression methods employed in our study, as both are central to our methodology.

Subsequently, we present Chapter 3 (*Related Work*), offering a broad overview of previous research in this domain. This includes studies on ECG classification, arrhythmia prediction, and applications of tensor decompositions such as BTD and BTTR. This chapter serves to position our work within the existing body of literature and highlight how our approach differs or builds upon prior studies.

Chapter 4 (*Methodology*) then describes the overall design of our approach. We outline how the data is preprocessed, how the decomposition and regression techniques are applied, how decomposition flows into regression and how features are selected, and how these steps are integrated into a complete analysis pipeline. This section is intended to give the reader a clear conceptual understanding of our strategy before implementation-specific details are introduced.

Following this, Chapter 5 (*Implementation*) provides a more technical and detailed description of how the methodology was realised in practice. This includes choices related to software, libraries, parameter settings, computational resources, and any challenges encountered during development.

Then in Chapter 6 (*Results and Evaluation*), we present the outcomes of our experiments. Quantitative performance metrics, such as classification accuracy or prediction error, can be used to assess the effectiveness of our approach. We also include visualisations and qualitative interpretations where relevant.

Next, Chapter 7 (*Discussion*) reflects on these results. We explore the implications of our findings, address potential limitations, and suggest possible explanations for any unexpected behaviour. Lastly, we talk about all the lessons that were learned in this work.

Finally, Chapter 8 (*Conclusion*) summarises the main findings of this work. We revisit the original objectives and highlight the key results that emerged from our methodology. This chapter also reflects on the extent to which these objectives were achieved, and briefly restates the effectiveness and limitations of the proposed approach. In addition, we suggest possible directions for future research, including improvements to the methodology and alternative decomposition or regression strategies.

# Chapter 2

# Background

This chapter is dedicated to reviewing the foundational concepts necessary for understanding the subsequent chapters. It mainly focuses on the core methodologies employed by BTTR, namely decomposition and regression. In this chapter, we delve into the mathematical concepts behind these techniques to ensure a comprehensive understanding. Additionally, we briefly discuss some basic ECG knowledge, although we didn't explore this in detail, as it is not essential for this study.

## 2.1 Tensor Decomposition

Tensor decomposition is a core concept in BTTR. There are various types of tensor decomposition methods, but in this thesis, we focus primarily on CP Decomposition (CPD), commonly referred to as CP decomposition. This technique can be seen as a special and simplified case of Block Term Decomposition (BTD), which is the general decomposition method used in the Block Term Tensor Regression (BTTR) model that we study in this work. We start by explaining CP Decomposition, then we explain the broader concept of BTD.

### 2.1.1 CP Decomposition

In essence, CANDECOMP/PARAFAC (CP) decomposition approximates a high-dimensional input tensor by expressing it as a sum of $R$ rank-1 tensors. Each of these rank-1 tensors identifies and represents a fundamental pattern in the data, also referred to as a latent component. Intuitively, each component captures a distinct structure or motif that's repeatedly observed across the dataset.

This decomposition not only reveals what patterns exist but also quantifies how strongly each pattern is expressed across all dimensions of the tensor. For example, in our case, the original tensor has dimensions $samples \times time(Hz) \times leads$. After decomposition, the factor matrices allow us to interpret which patients express each pattern the most, when in time (e.g., early or late in the ECG signal) a pattern occurs, and in which leads the pattern is most prominent.

For instance, the CP model might show that a certain latent pattern appears mostly at the start of an ECG, is dominant in a specific group of leads, and is strongly present in a particular subset of patients. This joint interpretation across all modes enables a rich and interpretable understanding of the hidden structure in the dataset.

To give a more formal definition: A rank-1 tensor can be constructed as the outer product of N vectors, where N is the dimension of the data. In our case, we form such a tensor using three vectors corresponding to samples, time, and leads. Each vector captures the contribution or intensity of a shared latent pattern across one of these dimensions.

The samples vector has a length equal to the number of ECG recordings or instances (denoted as s). Each element indicates the degree to which the underlying latent pattern is present in that particular ECG sample.

The time vector has a length equal to the number of time points in each recording. For example, with a 10-second recording sampled at 500 Hz, this results in a vector of length 5000. Each value reflects how strongly the latent pattern is expressed at that specific moment in time.

The leads vector corresponds to the number of ECG leads, with each entry indicating how prominently the latent pattern is manifested in each lead.

The outer product of these three vectors yields a rank-1 tensor, which captures a separable component of the overall ECG signal in terms of samples, time, and leads. A sum of two such tensors gives a rank-2 approximation (two patterns). The original data tensor can be considered to have very high or full rank, as it contains many different patterns. In other words, you would need almost infinite Rank-1 tensors in order to get the original tensor exactly.

The goal of CP decomposition is to find a rank-$R$ tensor, formed by the sum of $R$ rank-1 tensors, that approximates the original tensor as closely as possible, using only R*N vectors, see Figure 2.1. This reduction enables us to capture the essential latent structure of the data in a compact and interpretable form.

Let us now proceed to the formal definition of the CP decomposition.

$$\mathcal{X} \approx \sum_{r=1}^{R} \lambda_r \mathbf{a}_r \otimes \mathbf{b}_r \otimes \mathbf{c}_r \tag{2.1}$$

**Explanation of symbols:**

- $\mathcal{X}$: the original third-order tensor, typically of size $P \times T \times L$ (e.g., patients $\times$ time $\times$ leads).
- $R$: the rank of the decomposition, i.e., the number of components (patterns).
- $\lambda_r$: scalar weight associated with the $r$-th component (optional, sometimes absorbed into one of the vectors).
- $\boldsymbol{a}_r \in \mathbb{R}^P$: vector for the first mode (e.g., patients) of component $r$.
- $\boldsymbol{b}_r \in \mathbb{R}^T$: vector for the second mode (e.g., time) of component $r$.
- $\boldsymbol{c}_r \in \mathbb{R}^L$: vector for the third mode (e.g., leads) of component $r$.
- $\otimes$: outer product operator, producing a rank-1 tensor from three vectors.

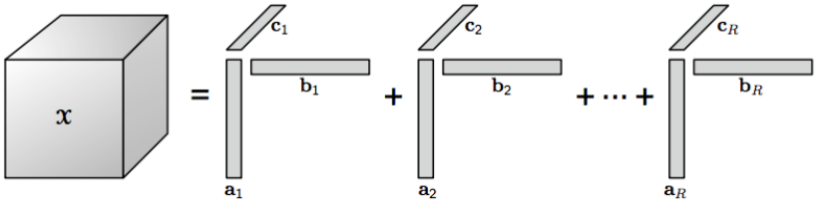


**Figure 2.1:** A mathematical visualisation of a CP decomposition. [12]

To compute this decomposition in practice, one commonly used method is **Alternating Least Squares (ALS)**. This is an iterative optimisation algorithm that updates one factor matrix at a time while keeping the others fixed. More specifically, ALS cycles through each mode (e.g., samples, time, leads) and solves a least-squares problem to update the corresponding factor matrix, given the current estimates of the other modes.

Intuitively, ALS tries to minimise the reconstruction error between the original tensor $\mathcal{X}$ and its CP approximation by refining the factor matrices iteratively. Despite its simplicity and ease of implementation, ALS can converge slowly and may get stuck in local minima. However, it remains one of the most widely used algorithms for decomposition due to its interpretability and scalability to large datasets.

The output consists of a set of $R$ rank-1 tensors, or equivalently, $R$ sets of $N$ vectors—one set for each latent factor. These tensors can be reorganised into *factor matrices*, one for each mode (dimension) of the original data tensor. In our case, the data has three modes: **ECG samples** $\times$ **Time** $\times$ **Leads**.

To construct the *samples factor matrix*, we stack all $R$ sample vectors (one per latent factor) into a matrix of shape $s \times R$, where $s$ is the number of ECG samples. Each row corresponds to a sample, and each column corresponds to a latent factor. The entries in this matrix indicate the degree to which each ECG sample expresses each of the $R$ latent patterns.

Similarly, we can form factor matrices for the *time* and *lead* dimensions, capturing how each time point and each lead contributes to the various latent factors.

### 2.1.2 Block Term Decomposition

Block Term Decomposition (BTD) is a generalisation of the CP decomposition where a low-rank Tucker block replaces each rank-1 term. This allows more expressive modelling of tensor data by capturing local structures within each term. [13]

For a third-order tensor $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$, the BTD can be written as:

$$\mathcal{T} \approx \sum_{r=1}^{R} \mathcal{S}_r \times_1 \mathbf{U}_r \times_2 \mathbf{V}_r \times_3 \mathbf{W}_r,$$

where: - $\mathcal{S}_r \in \mathbb{R}^{L_r \times M_r \times N_r}$ is a small core tensor of rank-$(L_r, M_r, N_r)$, - $\mathbf{U}_r \in \mathbb{R}^{I \times L_r}$, $\mathbf{V}_r \in \mathbb{R}^{J \times M_r}$, and $\mathbf{W}_r \in \mathbb{R}^{K \times N_r}$ are the factor matrices for the $r$-th block. [13]

Each term in the sum represents a localised Tucker decomposition, allowing the model to capture complex and heterogeneous patterns within the data. In contrast to CP, where each term is strictly rank-1, BTD supports a richer representation by using a multilinear low-rank block per term.

Figure 2.2 visualises this concept, showing how the tensor $\mathcal{T}$ is approximated by a sum of Tucker blocks built from factor matrices and core tensors.

In this thesis, BTD is introduced for completeness, as it represents a more general framework encompassing CP as a special case. However, we primarily focus on CP-based models such as mPSTD, ACE, and ACCoS due to their balance between expressiveness and computational feasibility.

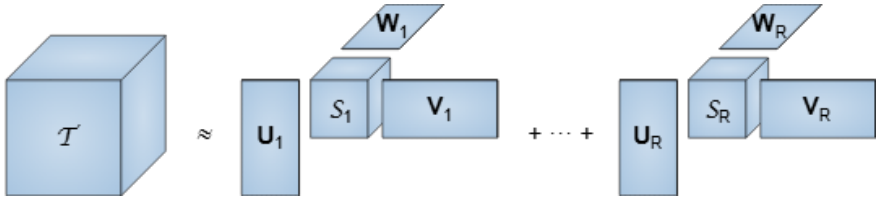


**Figure 2.2:** A mathematical visualisation of Block Term Decomposition [14]

## 2.2 Regression models

In this section, we provide a brief overview of the regression models used in this work. These include logistic regression, random forest, and XGBoost. While all of them can be used for classification tasks, they differ significantly in how they model the data and handle complexity.

### 2.2.1 Logistic Regression

Logistic regression is a linear model commonly used for binary classification. It estimates the probability that a given input belongs to a particular class using the logistic (sigmoid) function:

$$P(y = 1 \mid \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x} - b)}$$

where $\mathbf{w}$ is a weight vector and $b$ is a bias term. Logistic regression is simple, interpretable, and works well when the relationship between the features and the target is approximately linear. [15]

### 2.2.2 Random Forest

Random forest is an ensemble learning method that builds multiple decision trees during training and outputs the class that is the mode of the classes predicted by individual trees. It introduces randomness by bootstrapping data samples and by selecting random subsets of features at each split. Compared to logistic regression, it can capture more complex patterns and interactions in the data, but it is less interpretable. [16]

### 2.2.3 XGBoost

XGBoost (Extreme Gradient Boosting) is another ensemble method, based on gradient boosting of decision trees. Unlike random forest, which builds trees independently, XGBoost builds trees sequentially to correct the errors of previous trees. It includes various optimisations such as regularization and efficient handling of missing data. XGBoost typically achieves higher accuracy than random forest in many tasks but can be more sensitive to hyperparameters. [17]

## 2.3 Block Term Tensor Regression

As mentioned before, BTTR is a combination of a tensor decomposition and a regression technique. As for the decomposition, we specifically use CP Decomposition as a simplified version of Block Term Decomposition (BTD), but the same concepts apply to a complete BTD.

The decomposition output is structured into $N$ factor matrices, one for each mode of the data tensor. In our case, we focus on the *ECG samples factor matrix*, as it encodes the relationship between individual samples and the latent factors. This matrix effectively serves as a feature representation of the samples and is therefore suitable for use in regression tasks.

The other two factor matrices, corresponding to *time* and *leads*, do not provide sample features and are thus not directly useful for regression. However, they play an important role in the interpretability of the latent components, as they reveal how each latent pattern is distributed across time and leads.

## 2.4 Electrocardiograms (ECGs)

An electrocardiogram (ECG) is a signal that measures the electrical activity of the heart over time. It is used to detect and monitor different heart conditions. A typical ECG signal, like in

Fig 2.3, consists of repeating patterns called heartbeats or rhythms. Each rhythm contains a set of characteristic waves: the P wave, QRS complex, and T wave. [18]

- **P wave:** represents the atrial depolarisation, or in simple terms, the electrical activity that causes the atria (upper heart chambers) to contract.
- **QRS complex:** shows the depolarisation of the ventricles, the main pumping chambers of the heart. This is usually the most prominent part of the ECG.
- **T wave:** reflects the repolarisation of the ventricles, meaning the recovery phase after a contraction. [18]

Each of these waves has a typical shape and timing, and changes in them can indicate heart problems. There are different types of rhythms, and not all are healthy. For example, in a normal sinus rhythm, all the waves appear in a regular pattern. In contrast, arrhythmias are irregular rhythms.
In **atrial fibrillation** for example, the rhythm becomes chaotic and the P waves appear to be absent. [19]

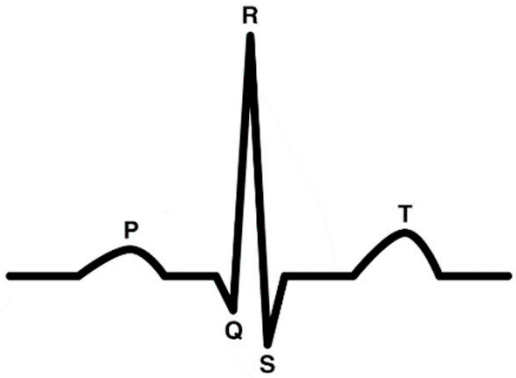


**Figure 2.3:** A healthy ECG waveform. [20]

# Chapter 3

# Related work

This chapter explores various related research topics, specifically focusing on the classification and prediction of atrial fibrillation. Both aspects are of interest; as such, our investigation begins with classification before advancing to prediction. Although we didn't use most techniques discussed in the topics, and instead focus on Block Term Tensor Regression (BTTR), exploring these works can give us valuable insights and allow us to do interesting comparisons within the same research domain. Additionally, we examine BTTR-related works, which may not necessarily incorporate ECG data. We do this in order to get a better view to what it is capable of.

## 3.1 Early AF Detection Using Pattern Recognition

The first work we describe is named: *"Pattern recognition and automatic identification of early-stage atrial fibrillation, by Wu et al. (2020)"* [1]. This work presents a method for the automatic detection of early-stage atrial fibrillation (AF) using short-duration, single-lead ECG signals. Their main goal is to enable out-of-hospital and real-time AF detection, especially targeting short and paroxysmal episodes that are often missed in clinical settings due to their unpredictable nature. To address this, the authors move away from traditional rhythm analysis methods that rely on long ECG recordings or multi-lead data. [1]

The proposed AI pipeline consists of several steps. First, RR intervals are extracted from the ECG using the Pan–Tompkins algorithm, which means that they extract one heartbeat out of the 10-second sample. These heartbeats are then processed using Intrinsic Time-scale Decomposition (ITD), a data-driven method that decomposes the signal without requiring predefined basis functions. From the ITD output, three handcrafted features are computed: intrinsic energy, intrinsic frequency entropy, and intrinsic time entropy. These features aim to capture the irregularity and complexity of AF. A support vector machine (SVM) is trained on these features to classify between AF and sinus rhythm. [1]

The model achieves high performance on PhysioNet data, reporting 95% accuracy, 96% specificity, and 93% sensitivity. A key strength of the approach is its ability to operate on very short ECG segments of just 1 rhythm, as well as only needing 1 lead, making it suitable for simple wearable or mobile applications. However, the reliance on handcrafted features may limit adaptability to broader datasets. Furthermore, the use of only single-lead ECG input, while practical, could restrict diagnostic richness in more complex clinical cases. What is interesting is the extraction of RR intervals, resulting in a segmentation into heartbeats. This is not something we did, as rhythms span over multiple beats and it might be valuable to see the complete context in order to correctly classify AF. However, the big downside of not doing this segmentation is the amount of data and the noise, making it more challenging for the model. [1]

Therefore, in this study, we aimed to utilise larger samples with multiple beats and 12 leads. This increases the complexity and presents a greater challenge. Nonetheless, this approach holds the potential to result in significant medical insights in the future.

## 3.2 Deep Transfer Learning for ECG Classification

The next work, named: *"ECG Heartbeat Classification: A Deep Transferable Representation, by Kachuee et al. (2018)"* [3], proposes a deep learning framework for the classification of ECG heartbeats and the transfer of learned representations to other related cardiovascular tasks. The primary objective of their work consists of two parts: first, to achieve accurate heartbeat classification into arrhythmia categories, following the AAMI EC57 standard. Second, to explore whether the representations learned from this task can be reused for a different but related problem: myocardial infarction (MI) detection. This could be a solution to limited labelled data in many clinical tasks, offering a transferable model that could generalise across multiple cardiac conditions.

The authors design a deep residual convolutional neural network trained on beats extracted from the MIT-BIH Arrhythmia Database. Their preprocessing pipeline includes beat segmentation using R-peak detection, amplitude normalisation, and fixed-length beat windowing. After training on arrhythmia classification, they extract deep feature representations from the final convolutional layer and reuse them to train a lightweight MI classifier using data from the PTB Diagnostic ECG Database. Only the final layers of the MI classifier are retrained, while the rest of the network is frozen, demonstrating a transfer learning setup.

Results show strong performance: the arrhythmia classifier reaches an average accuracy of 93.4%, and the transferred model for MI classification achieves 95.9% accuracy. One limitation is that while the representations appear generalisable between arrhythmia and MI tasks, the system's dependence on large training data for the source task may still be a barrier in low-resource applications. Additionally, again the method does not yet incorporate multi-lead information, which could enhance diagnostic performance in more complex clinical settings.

Although transfer learning is a very useful and interesting concept, it's not directly relevant to our work, as it is a big research topic on itself, and not our focus. However, the classification is relevant. The researchers chose for a deep learning methodology, widely recognised for its effectiveness, and achieved strong results. However, our focus is to investigate whether BTTR might serve as an effective approach to address this problem. Another notable difference is that this involves a heartbeat classification issue, as opposed to a rhythm classification issue. Thus, there is a partial comparability to the previously addressed subject, where segmentation led to working with individual beats.

## 3.3 Deep Learning for ECG Classification

The work *"Deep Learning for ECG Classification, by B Pyakillya et al."* [4] also addresses the challenge of creating an effective arrhythmia classifier, using only a single short-lead ECG. The authors highlight the limitations of traditional machine learning algorithms that rely on heuristic hand-crafted or engineered features. They mention that a key problem with this approach is the potential failure to identify the most appropriate features from ECG signals for accurate classification.

To overcome these limitations, they propose using deep learning architectures. In their methodology, the initial convolutional layers function as feature extractors, automatically learning relevant representations from the raw ECG data. These convolutional layers are followed by fully connected layers, which perform the final classification of the ECG signals.

The study utilised a dataset of ECG recordings categorised into four classes: normal sinus rhythm, arrhythmic, other rhythm, and noisy. To address the issue of imbalanced data, where

the majority of recordings belonged to the normal sinus rhythm category, they performed data augmentation techniques. These techniques included multiplying existing ECGs by shifting time values and unifying ECG lengths by duplicating time-series values. Standard preprocessing, such as subtracting the mean and dividing by the standard deviation, was also applied to the time-series data. The deep learning architecture consisted of 1D convolutional layers, max-pooling, dropout, Global Average Pooling, and fully connected layers, culminating in a softmax layer for classification.

The results of their experiments showed that the proposed deep learning solution could effectively classify ECG signals, even with unstructured and unbalanced data. They achieved a best validation accuracy of approximately 86

## 3.4 AF detection on patients during sinus rhythm

This next work, named: *"An artificial intelligence-enabled ECG algorithm for the identification of patients with atrial fibrillation during sinus rhythm: a retrospective analysis of outcome prediction, by Attia et al. (2019)"* [5] has been very eye-opening, as it shows that AF signs can be found in sinus rhythms, hence we can predict AF from sinus rhythms.

This work was performed by researchers from the Mayo Clinic in Rochester, where they developed an artificial intelligence-enabled electrocardiograph (ECG) algorithm to identify patients with atrial fibrillation during periods of normal sinus rhythm. Their motivation behind this work is the same as ours, being the clinical challenge that AF is often asymptomatic and unpredictable, making it difficult to detect with regular ECG screenings. Their goal was to provide a non-invasive, cost-effective, and scalable method to improve AF diagnosis, particularly in patients at risk for stroke due to undetected arrhythmias.

The study used a convolutional neural network (CNN) architecture on standard 10-second, 12-lead ECG recordings from a large private dataset collected at the Mayo Clinic. The dataset consists over 649,000 ECGs from more than 180,000 patients, split into training, validation, and testing sets. The network was trained to detect subtle structural changes in the atria indicative of a history of AF, despite the ECG showing normal sinus rhythm at the time of recording. The methodology included using eight out of twelve independent ECG leads and applying deep learning techniques like residual blocks and ReLU activations for robust feature extraction.

The results showed promising performance: using only a single ECG, the model achieved an AUC of 0.87, with approximately 79% accuracy, sensitivity, and specificity. Incorporating multiple ECGs per patient increased the AUC to 0.90, showing that these additional leads include additional information. These results significantly outperform traditional methods like CHA2DS2-VASc scoring or short-term Holter monitoring in terms of predictive accuracy and convenience. However, despite strong overall performance, the model's F1 score remained modest (around 39–45%), partly due to class imbalance and the inherent difficulty of the prediction task.

This research is very promising for the field of medical research. However, the issue of explainability remains, as the use of a neural network results in it operating as a 'black box', thus providing limited insights in what the model has learned. Understanding what the neural network identifies in sinus rhythms that indicate signs of atrial fibrillation (AF) would be very valuable. Unfortunately, the neural network currently lacks this capacity. Such challenges are the reason for the growing importance of explainable AI within machine learning, which is one of the reasons we tried using BTTR instead of Neural Networks. Another challenge they faced, which we faced as well, is the possibility that some false positives actually reflect undiagnosed AF, highlighting the challenge of correct labelling in such studies. Nonetheless, they still achieve high accuracy.

## 3.5 AF detection using sinus rhythm ECGs

A similar study is the following: *"Detecting Paroxysmal Atrial Fibrillation from an Electrocardiogram in Sinus Rhythm"*, conducted by Gruwez et al. (2023) [6]. They had the same motivation and goal as the previous work by the Mayo Clinic, which was the detection of paroxysmal AF from sinus rhythm recordings. Thereby contributing to stroke prevention through earlier diagnosis. They approached the problem the same way, using all leads, but used different data. Being raw ECG data from Hospital Oost-Limburg (ZOL) in Genk, Belgium, where they collected over 494,000 ECGs in SR from 142,310 patients. The data was split into training (70%), validation (10%), and testing (20%) sets. As proposed by Attia et al. the algorithm was built using convolutional neural networks, implemented in Keras and trained on a Spark cluster. The input data consisted of eight independent ECG leads sampled at 500 Hz.

When tested on the first ECG in SR of each patient, the model achieved an AUROC of 0.87 and an AUPRC of 0.48, with accuracy, sensitivity, and specificity all around 78%. These results were comparable to the original Attia model, despite being trained on different data. Furthermore, Gruwez et al. performed an external validation using a dataset from Ziekenhuis Maas en Kempen (ZMK), including over 70,000 ECGs from 26,000 patients. The model maintained a strong performance (AUROC 0.86), showing generalisation across institutions. However, in populations with lower AF prevalence, metrics like AUPRC and F1 score decreased, highlighting limitations in positive predictive value under screening conditions.

A key strength of this work is its rigorous validation in a different healthcare system and dataset, addressing concerns about generalisability, a common issue in AI healthcare applications. Nonetheless, the study faced the same limitations related to retrospective labelling based solely on ECG database records, which may have misclassified some AF cases. Also, full interpretability of the DNN remains unresolved, as Neural Networks act as black boxes. Still, the findings again confirm that AI can identify subtle atrial fibrillation signs in ECGs and may assist clinics and smartwatches in diagnosing people with AF in normal circumstances.

## 3.6 Block-Term Tensor Regression for Neural Signal Prediction

We now describe *"Single finger trajectory prediction from intracranial brain activity using Block-Term Tensor Regression with fast and automatic component extraction, by Faes et al."* as a first work regarding decomposition. [21]
Trajectory prediction is a critical aspect of human-computer interaction (HCI) and neuroprosthetics, with a growing research focusing on predicting hand and finger movement trajectories. Single-finger trajectory prediction, in particular, has gained attention due to its potential applications in enhancing the performance of assistive devices, prosthetics, and advanced HCI systems.

Previous efforts in decoding finger movement trajectories from ECoG signals have primarily relied on conventional machine learning techniques such as sparse and linear regression models, Gaussian processes, CNNs, Random Forests, and LSTMs.

While these methods have good results, they often ignore the intrinsic multi-dimensional structure of ECoG data. Recent developments in multiway tensor-based approaches, such as Tucker, CP decompositions and Block Term decomposition, have attempted to address this by preserving the spatial dimensions of brain signals. Notably, Higher-Order Partial Least Squares (HOPLS) regression has showed improved prediction accuracy over traditional techniques but remains limited by the requirement of manually selected model parameters and fixed tensor ranks.

To overcome these limitations, some methods have incorporated automatic rank estimation through Bayesian learning, pruning strategies, or information-theoretic criteria like MDL. How-

ever, these still often require high computational costs. This thesis builds on and extends these methods by introducing the Block-Term Tensor Regression (BTTR) framework equipped with automatic component extraction and rank selection (ACE/ACCoS), enabling fast, robust, and accurate decoding of single-finger trajectories from intracranial recordings, and achieving improved prediction performance over both conventional and state-of-the-art tensor-based approaches.

The Block-Term Tensor Regression (BTTR) framework achieved the highest Pearson correlation scores for predicting single-finger trajectories in the BCI Competition IV dataset, outperforming linear regression, Random Forests, CNNs, LSTMs, and HOPLS. It trained significantly faster than HOPLS, making it more suitable for time-sensitive applications. The use of structured tensor components (via ACE and ACCoS) provided interpretability, revealing the spatiotemporal-frequency components contributing to motor decoding. ACE reduced model complexity by removing irrelevant components, while ACCoS improved predictive power by selecting the most relevant ones. Regularisation through marginalisation helped prevent overfitting, especially with a small number of components. Overall, BTTR proved to be a robust, efficient, and interpretable method for decoding motor trajectories from ECoG data, highlighting the benefits of exploiting the multiway structure of ECoG signals. The component selection strategy retained relevant patterns, offering better performance than black-box models like deep neural networks without sacrificing efficiency or transparency, making it a promising approach for future brain–computer interface systems.

## 3.7 Generalised Tensor Decomposition for Neural Data

The paper *"Generalised Canonical Polyadic Tensor Decomposition, by Hong, Kolda, and Duersch (2020)"* [22] represents a significant contribution that enhances the applicability of the classic CP decomposition. The primary objective of this work is the development of a Generalised Canonical Polyadic (GCP) tensor decomposition method. What makes this approach interesting is its ability to perform CP decomposition using a variety of "loss functions", instead of just reconstruction error. This is particularly valuable as it enables the application of CP decomposition to a wider range of data types. The authors provide a robust framework for calculating gradients and handling missing data, thereby facilitating the use of standard optimisation techniques to fit the model.

The methodology of this paper extends the familiar CP decomposition by generalising its objective function. This allows for the fitting of the CP model to data according to various statistically motivated loss functions. Optimisation is performed via a gradient-based approach, which offers a more flexible alternative compared to the commonly used Alternating Least Squares (ALS) method for CP. To demonstrate the versatility of GCP, the authors apply it to several real-world scenarios. For our work, a particularly relevant example is their analysis of *neural activity in mice*. This specific dataset, derived from calcium imaging of the mouse prefrontal cortex, is structured as a third-order tensor with dimensions such as 'neuron × time × trial' (for instance, 282 neurons across 111 time points over 300 trials). The decomposition of this data gives interpretable patterns and factors that reveal neural assemblies, temporal dynamics within individual trials, and how these patterns vary across different trials.

Ultimately, this work strongly highlights the efficiency of CP decomposition for extracting meaningful, latent patterns from complex, multi-dimensional biological data. This is the case for both neural signals, as shown in their paper, and for our own ECG data. Furthermore, it also really highlights the interpretability of decomposition models like CP.

## 3.8 Other Applications of CP Decomposition

### 3.8.1 Chemistry and Spectroscopy

In analytical chemistry, labs often use spectroscopy to figure out what's inside a substance. This involves shining different kinds of light on a material and recording how it responds. Here, CP decomposition is used to identify and measure the specific chemical compounds, or "ingredients," within such a mixture. It works by taking the combined signals from the mixture and separating them into distinct patterns for each individual component. This means you can accurately analyse these substances, even if you don't know exactly what's in the mix beforehand. [23]

### 3.8.2 Recommender Systems and Data Mining

CP decomposition is also used in recommendation systems (like those suggesting movies or products) and for general data analysis. In these fields, the data often represents interactions across several different categories. For example, which user likes which genre of movies, at what day of the week. CP is effective at understanding all these relationships at the same time, seeing how various factors influence each other simultaneously. This capability ultimately leads to more accurate recommendations and a better grasp of complex user behaviour or system dynamics. [24]

### 3.8.3 Signal Processing and Blind Source Separation

In signal processing, CP decomposition offers significant help with a challenge called Blind Source Separation (BSS). For example, you might have recordings where several signals are all mixed together, like different people speaking in one audio file, or various biological activities picked up by a single sensor. BSS is the technique for separating these combined signals to recover their original signals. [25]

# Chapter 4

# Methodology

## 4.1 Research Approach

This study follows an experimental approach to assess how effective a simple BTTR algorithm can be for ECG classification and arrhythmia prediction.

To structure our investigation, we first focused on a diagnostic classification task, as this is conceptually simpler and allowed for systematic experimentation with preprocessing, decomposition, and model selection. Starting with classification enabled us to explore what worked best in terms of signal preparation and parameter choices. Once we identified a well-performing pipeline, we adapted it for the prognostic prediction task, which primarily involved changing the sampling strategy to match the temporal setup required for forecasting rather than diagnosing. The rest of the pipeline remained largely unchanged, benefiting from the insights gained during the classification phase.

The methodology itself consists of several key stages: selecting appropriate ECG data, applying signal preprocessing, and using CANDECOMP/PARAFAC (CP) tensor decomposition to extract meaningful patterns from the data. We evaluate the influence of various decomposition parameters and rank selection strategies.

Following tensor decomposition, automated feature selection and preprocessing techniques are applied to the sample factor matrix to reduce dimensionality and enhance signal quality. Several machine learning classification algorithms are then tested and compared in terms of their predictive performance. We systematically explore different hyperparameter configurations for these models using a grid search.

Finally, performance is assessed using standard accuracy metrics, and visualisation techniques are used to interpret the decomposition results and gain insights into the model's behaviour. This approach ensures a comprehensive investigation of each component involved in ECG classification and prediction using tensor-based representations.

## 4.2 BTTR Overview

In this section, we give a broad overview of the full pipeline used in this work, referred to as BTTR, which stands for Block Term Tensor Regression [21]. Each part of this method is explored in more detail in the sections that follow.

BTTR combines two key steps: first, an unsupervised decomposition of the ECG signals to extract useful patterns, and second, a regression or classification model that uses those patterns to predict the heart rhythm.

We start with a large set of ECG recordings. These are first preprocessed to clean up noise and prepare them for further analysis. Once the data is ready, we apply a tensor decomposition technique that automatically finds a number of recurring patterns across all ECGs. The goal is to represent each ECG as a combination of these shared patterns. This step is completely unsupervised: the model does not use any label information while finding these patterns.

Once the patterns are extracted, we move to the second stage. Now that each ECG has been reduced to a set of pattern values, we look at how these relate to the known heart rhythm labels (e.g. AF or SR). We train a supervised model on these values to learn the relationship between patterns and rhythm type.

In the end, this allows us to take a new, unseen ECG, reduce it to the same set of pattern values, and use the trained model to predict which heart rhythm it most likely belongs to.

This technique is powerful because it creates a low-dimensional, interpretable representation of the ECGs while still achieving strong classification performance. It works fully unsupervised at first, and only later uses the labels to map patterns to outcomes, which makes it especially interesting for biomedical data where interpretability and structure matter.

## 4.3 Dataset

### 4.3.1 PTB-XL ECG

The data that was used in this work was the PTB-XL ECG database from PhysioNet, a large publicly available electrocardiography dataset. [2, 7, 8]

The PTB-XL ECG dataset is one of the largest publicly available ECG datasets, developed and published by a collaborative team of researchers affiliated with institutions in Germany and the United Kingdom. It consists of 21,799 ECG recordings, each 10 seconds long and sampled at 500 Hz. Each recording is a 12-lead ECG, meaning all standard leads were recorded (I, II, III, aVL, aVR, aVF, V1–V6), providing the maximum amount of information. These raw waveform ECGs were collected between October 1989 and June 1996 at the Physikalisch-Technische Bundesanstalt (PTB) in Germany, using Schiller AG devices.

Each recording was annotated by up to two cardiologists, who labelled the rhythms using so-called ECG statements. More specifically, there are 71 different possible ECG statements conforming to the SCP-ECG standard. These include diagnostic, form, and rhythm statements, covering all essential information required for interpretation.

In addition, each record includes pseudo-anonymised metadata, such as age, sex, height, and weight. Basic information about the recording itself, such as the date, recording site, and device type is also provided. Furthermore, as stated by PhysioNet, the dataset includes extensive supplementary data, including demographic details, infarction characteristics, likelihoods for diagnostic ECG statements, and annotated signal properties.

The PTB-XL ECG dataset is the most widely used dataset by researchers for developing machine learning algorithms in the field of ECG analysis. The only real alternative would be collaborating directly with a hospital to access clinical data, which is often difficult due to privacy concerns and limited availability. One of the main advantages of using a widely adopted dataset like PTB-XL, used in a lot of related works, is that it allows for easier comparison between different algorithms. This made it an obvious and straightforward choice for our project.

### 4.3.2 Dataset structure

The dataset is thoroughly structured into multiple files and folders. At the top level, we can first of all find the "ptbxl_database.csv" file and the "scp_statements.csv" file.

The database file contains all relevant information for each ECG recording, including the assigned SCP-ECG codes like AFIB (atrial fibrillation) and SR (sinus rhythm). Each ECG

recording is typically associated with multiple such codes, where each code corresponds to a diagnostic statement defined in the scp_statements.csv file.
These codes and statements follow the SCP-ECG (Standard Communications Protocol for Computer-Assisted Electrocardiography) standard [26], a European specification for storing and exchanging ECG data. This standard defines how various components, including waveform signals, patient metadata, and diagnostic information, are formatted and encoded.

The diagnostic statements describe the medical conditions or abnormalities identified in the ECG. They are encoded using short SCP-ECG codes such as SR for sinus rhythm or AFIB for atrial fibrillation. Since a single ECG can exhibit multiple rhythm types or abnormalities, multiple diagnostic statements may be assigned to a single recording. In addition to these coded labels, each recording is also accompanied by a human-readable description of the findings.

In total, 71 unique statements (and hence codes) are defined in the scp statements file.
These statements contain information like the diagnostic and its discription, the rhythm type, diagnostic class and subclass and so on.
In addition to the CSV metadata files, the dataset also includes the actual ECG waveform recordings. These are provided in two separate directories: records500 and records100. The records500 directory contains the original recordings sampled at 500 Hz, while records100 contains a downsampled version of the same data at 100 Hz.

Each of these directories is further divided into 22 subdirectories named 00000, 01000, 02000, ..., up to 21000. Each subdirectory contains up to 1,000 ECG recordings, organized to efficiently manage the large dataset size. Every ECG recording consists of two files:

A id_lr.dat file, which contains the raw 12-lead ECG signal in binary format.
A corresponding id_lr.hea file, which is a header file storing metadata such as sampling frequency, lead configuration, and signal calibration parameters for each channel. These are technical details that we did not use.

Finally, a Python script is provided to facilitate the extraction of all ECG recordings from the dataset's folder structure.

### 4.3.3 Used data

As is clear by now, the dataset is very extensive, containing a large amount of metadata and detailed rhythm annotations. However, not all of this information was relevant for our work. Personal metadata such as age and sex is often included in predictive models for arrhythmias like atrial fibrillation (AF), since men are more prone to AF and its prevalence increases with age. These features can therefore contribute to predictive performance.

Nevertheless, the aim of our project was to rely exclusively on the ECG signals themselves. The motivation behind this choice was to develop an algorithm capable of accurately detecting or predicting AF purely from ECG data, which directly reflects the electrical activity of the heart, rather than relying on indirect demographic indicators like age or sex, which are not diagnostic on their own.

That said, we did require some metadata. The metadata file also included fields such as patient ID, ECG ID, and recording time. Furthermore, we also needed the SCP-ECG codes, as these form our labels. For our purposes, we focused exclusively on the SCP-ECG codes relevant to our classification and prediction task: SR, indicating a healthy sinus rhythm, and AFIB, indicating atrial fibrillation. Only recordings labelled with one of these two codes were included in our dataset. We did not make use of the detailed diagnostic statements or textual descriptions, our only requirement was to identify which recordings corresponded to SR and which to AFIB. In total, out of 21799 recordings, 16782 recordings were labelled with SR, and 1587 with AFIB.

## 4.4 Diagnostic Classification vs. Prognostic Prediction

As mentioned, we address two related but fundamentally different tasks: diagnostic classification and prognostic prediction. While both involve processing ECG signals to detect atrial fibrillation (AF), the nature of the input data and the clinical goal differ significantly.

The diagnostic classification task involves identifying whether a given ECG recording shows atrial fibrillation (AF) or a normal sinus rhythm (SR). In this task, the presence of AF is already relatively visible in the signal, and the objective is to correctly label it. This task is considered less complex, as AF and SR often show clear differences in rhythm and morphology.

In contrast, the prognostic prediction task requires identifying the future risk of AF from ECG recordings that currently show only healthy sinus rhythms. The goal is to detect subtle patterns or early indicators in seemingly normal ECGs that may suggest a patient is likely to develop AF in the near future. This task is significantly more challenging, as the signals do not show obvious signs of arrhythmia that are visible to the eye of a doctor. However, previous studies have shown that ECGs recorded days or weeks before an AF episode can already contain subtle signs of AF, which can be found using deep learning models.

Ultimately, both tasks reduce to the same fundamental goal: identifying patterns in ECG signals that can distinguish between two classes. In the case of diagnostic classification, these patterns separate recordings that visibly contain AF from those that do not. For prognostic prediction, the challenge is to detect more subtle patterns that differentiate between sinus rhythms from patients who will later develop AF and those who will not. Despite the difference in signal complexity, the underlying methodology remains the same, consisting of decomposing the signal, extracting informative features, and training classifiers to recognise these patterns.

This is why most of the development and experimentation could initially focus on the easier classification task and later be applied to prediction with different sampling. Since the experimentation was done in the context of classification, most chapters are explained from that perspective. However, you can keep in mind that the methodology applies to both tasks. Any differences, such as in sampling, are clearly mentioned.

## 4.5 Sampling

Now that we have collected our data from the PTB-XL ECG dataset, we applied targeted sampling to extract the relevant data for our two tasks. As previously mentioned, the initial filtering step involved discarding any ECG recordings that did not contain either an SR or AF label. From this point, the sampling process diverged depending on the specific task.

### 4.5.1 Task-Specific Data Labelling and Sampling Strategies

For the classification task, the sampling procedure is straightforward. We selected all recordings that were labelled with either SR (sinus rhythm) or AFIB (atrial fibrillation) codes. These served as the two target classes for binary classification: SR-labelled ECGs were assigned class 0, and AF-labelled ECGs were assigned class 1.

In contrast, the prediction task required a more complex sampling strategy. The objective here is to predict future AF onset based solely on sinus rhythm ECGs, meaning all input signals appear normal at the time of recording. We defined two classes:

Healthy (0): patients who did not have any AF recordings in the dataset. For this group, we collected all their SR-labelled ECGs.

At risk (1): patients who did have at least one AF-labelled recording. From these patients, we extracted only their SR-labelled ECGs recorded within one month prior to their earliest AF episode. This resulted in only 42 samples.

This sampling strategy ensures that both classes contain ECGs appearing in sinus rhythm, but only one class is known to precede an AF episode. This subtle distinction is central to the difficulty of the prognostic prediction task and reflects the real-world challenge of detecting risk based on apparently normal ECG signals.

An important thing to be aware of however, is that although we labelled certain patients as healthy based on the absence of AF-labelled ECGs in the dataset, it is important to acknowledge a potential source of bias. The fact that no AF episode was recorded does not necessarily mean that the patient never experienced atrial fibrillation. It is entirely possible that a patient had AF episodes that simply were not recorded during his visits at the hospital.

This introduces a limitation in the dataset: some ECGs that are labelled as healthy sinus rhythms may in reality belong to patients with undiagnosed or unrecorded AF. As a result, the distinction between healthy and at-risk patients may not be perfectly clean, and the "healthy" class may contain mislabelled samples. This type of misclassification is a form of label noise, and unfortunately, it cannot be fully resolved given the available data. It is an inherent limitation in many real-world clinical datasets and should be considered when interpreting the results and evaluating the model's predictive performance.

### 4.5.2 Class Balance, Sample Distribution and Sample Sizes

Now that we've filtered out all unusable ECGs and assigned the correct labels, we need to take a final sample to pass into the decomposition step. At this point, the dataset still contains too many ECGs, and we also want to control the ratio between class 0 and class 1.

Ideally, we aim for a balanced class distribution going into decomposition. This is important because our training set later on will also be balanced, and we want to preserve as much relevant data as possible for that stage. Depending on the test strategy, however, we sometimes prefer a more natural, real-world class distribution in the test set. This gives a better indication of how the model might perform in practice. In those cases, we can still apply undersampling later on, when we split the features into a training and testing set, to deal with the class imbalance during classification. Hence, even in this case, we choose for a balanced distribution into decomposition, because the test set needs the most amount of data.

For the prediction task, we are more limited: only 42 samples are available for class 1 (the risk class). To make the most of the data, we include a larger number of class 0 samples during the decomposition step. Later on, we choose a training distribution of 32/32 and a test distribution of 40/10 (class 0/class 1). This setup ensures that we always test on a more realistic, imbalanced distribution, as we want to evaluate the model on as much test data as possible.
The remaining class 1 samples, just 32 in total, are used for training. To keep the training set balanced, we also limit class 0 to 32 samples. This means the amount of training data for the prediction task is very small. However, one of the strengths of BTTR is its ability to uncover meaningful patterns, even when working with limited data.

Now that we had established our distribution strategy, we still needed to decide on the appropriate sample size for the classification task, where we have too much data. This process was largely experimental. We started with 100 samples per class, and gradually increased this up to 500 samples. As a final step, the labelled samples are shuffled to ensure a random distribution of classes throughout the dataset.

## 4.6 ECG Preprocessing

Before performing tensor decomposition, it is essential to preprocess the ECG signals to improve both data quality and decomposition performance. Raw ECGs collected from clinics often suffer from several issues, such as baseline drift, varying amplitude scales, high-frequency noise and excessively high sampling rate. without preprocessing, these problems can degrade the quality of the decomposition by introducing distortions or causing certain samples to dominate due

to scale differences. To address this, we applied a series of preprocessing steps that prevent samples from dominating the decomposition because of bigger amplitudes, reduce noise without discarding subtle predictive features and lower the sample rate without losing signal patterns. These operations not only improve signal comparability across samples, but also promote better low-rank approximation and pattern extraction during the decomposition stage. [27]

We experimented with a variety of preprocessing pipelines, each composed of multiple techniques, to investigate how ECG signals should be prepared in order to obtain a high-quality tensor decomposition. Specifically, we explored which aspects of the signals are most important: for instance, whether it is beneficial for all ECGs to fluctuate around zero, whether all signals should be scaled to a common range (i.e., having the same global minimum and maximum), or whether amplitude differences between signals should be preserved. [27]

To evaluate the effectiveness of each preprocessing pipeline, we used several strategies. First, we visualised the ECGs after each preprocessing step to validate that it does what we expect and to ensure we were not introducing excessive transformations. Next, we performed tensor decomposition and evaluated its quality using the reconstruction error and the Pearson correlation coefficient between the original and reconstructed signals, as a second measure of reconstruction error.

In addition, we evaluated how well the decomposed latent factors captured label-relevant information. In other words, how well both rhythm types are getting separated. For this, we computed both Pearson and Spearman correlation coefficients between the latent components and the labels, allowing us to quantify the relationship between the decomposition and the underlying ECG classes. Finally, we visualised the latent factors themselves by plotting the vectors that make up each rank-1 component of the decomposition. These evaluation metrics evaluate the quality of the decomposition, hence they are discussed in more detail in the decomposition section. However, we use them to evaluate the preprocessing as well, as the quality of the decomposition is strongly influenced by the chosen preprocessing strategy.

We first explain all the preprocessing techniques we tested (and possibly used), along with how each one affected the decomposition. After that, we'll walk through how we arrived at the final preprocessing pipeline.

### 4.6.1 ECG normalisation

There are a lot of normalisation techniques that we tested. The basic known normalisation is (global) min-max normalisation: [28]

$$\tilde{\mathcal{X}} = \frac{\mathcal{X} - \min(\mathcal{X})}{\max(\mathcal{X}) - \min(\mathcal{X})}$$

What happens with global min-max normalisation is that all values in the tensor are scaled to fall between 0 and 1, based on the global minimum and maximum across the entire tensor. The goal of this is to keep amplitude comparisons, since all samples are shifted and scaled by the same values, while bringing everything into the same scale, which is usually beneficial for machine learning models.

However, there's a catch. Even though all samples are technically in the [0, 1] range, their effective ranges can still vary a lot. Some samples might have signals that peak near 1 after normalisation, while others might only go up to 0.2, effectively placing them in a much smaller subrange like [0, 0.2]. This still leads to significant range differences between samples.

Even though this technique doesn't actually bring all ECGs to the same range in practice, it does preserve relative amplitude differences between samples. From a medical perspective, this can be useful, for example if one type of rhythm tends to have larger amplitudes than the other. But there are other things to keep in mind. First of all, it could be problematic to compare amplitudes across patients, as these can be skewed by using different machines or sensors that

are calibrated differently. The other major disadvantage is that, in practice, signals still lie in different ranges, which is often undesirable in machine learning.

In our experiments, we found that these differences indeed had a major impact on the decomposition. ECGs with large amplitudes (close to 1 after normalisation) ended up dominating the decomposition. As shown in Figure 4.1, most of the latent factors were shaped primarily by those signals. We saw this clearly when visualising the sample factor of the decomposition, where we see the outliers. The reconstruction error was also poor in this case, similar to what we observed before applying any normalisation, when amplitude variations were naturally present.

Weighing the advantage of preserving amplitude comparisons against the drawback of sensor-related variability in amplitudes could be a valid discussion. However, since we found that the decomposition struggles with large amplitude differences, we conclude that, in practice, it is better to bring all signals into the same range.

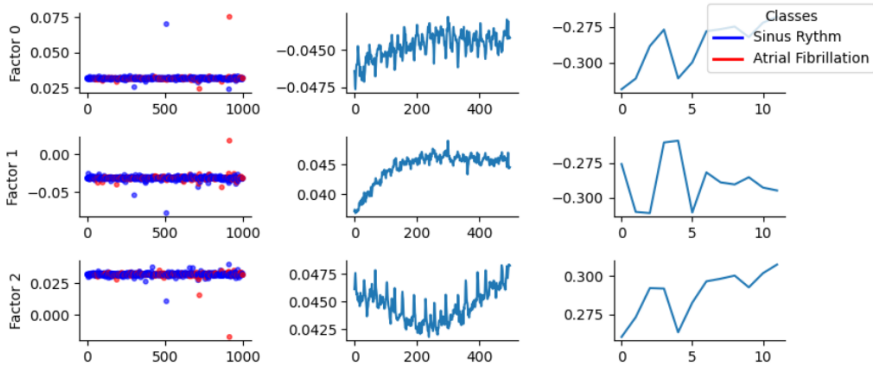


**Figure 4.1:** Decomposition influenced by outliers.
[29]

To address this issue, we applied per-signal normalisation. In this approach, the minimum and maximum are computed for each individual signal, and the normalisation is applied using those values. As a result, every signal is effectively scaled to lie between 0 and 1. This removed the large range differences we observed before, and we found that the decomposition results improved significantly. As shown in Figure 4.2, all samples now influence the decomposition. Although this method no longer allows for comparison of peak heights between signals, research showed that the overall shape or pattern of the signal often contains more meaningful information about the rhythm types. [30]

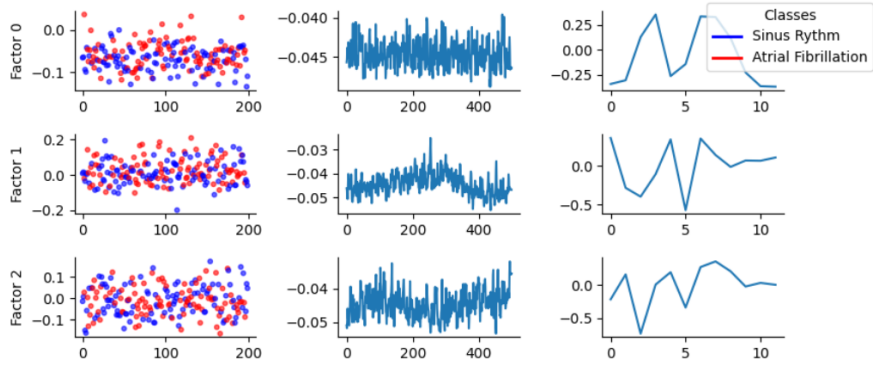


**Figure 4.2:** All samples contribute to the decomposition.
[29]

We also experimented with per-sample normalisation. In this case, we took the minimum and

maximum across all twelve leads of a single sample and used these values to normalise all the leads of that sample. The idea behind this was that there might be useful information in the relative peak heights across leads, and we hoped that the resulting range differences would not cause problems for the decomposition. However, just like with global normalisation, the results were poor.

How to interpret these visualisations is explained in Section 4.8.

### 4.6.2 Denoising

As mentioned, ECGs frequently suffer from various types of noise. First, there is baseline wander [31], a low-frequency shift in the ECG's baseline, typically caused by respiration, body movement, or poor contact between the electrodes and the skin.
Another common type is power line interference, which appears as a sinusoidal disturbance, usually around 50 to 60 Hz, introduced by nearby electrical devices or inadequate shielding.
Finally, ECGs can also be affected by very high-frequency noise, often resulting from electromagnetic interference, muscle contractions, or internal electronic noise from the recording equipment.
Such disturbances can obscure or distort clinically relevant features of the ECG, particularly the morphology of the P wave, QRS complex, and T wave, which are crucial for distinguishing between different rhythm types. [32]

To address this, we apply denoising to reduce or eliminate irrelevant fluctuations while preserving the original shape and timing of the waveform. This improves both visual interpretability and machine learning performance. Clean signals enable more accurate feature extraction and lead to better decomposition results. If noise dominates the signal, latent factor models may capture spurious patterns instead of the underlying rhythms, resulting in poor reconstruction and weak label correlation.

By reducing noise, we ensure that the decomposition captures the true structure of the ECG, making the subsequent analysis more reliable and meaningful.
The only real risk with denoising is removing too much of the signal. This is unlikely to be a problem for our classification task, as the differences between SR and AF are very obvious. However, for the prediction task, the signs of AF are often very subtle and not clearly visible to the eye. That's why, for this task, we are careful not to remove too much of the high frequencies.

Now, in order to denoise, we use three functions. The first one removes baseline wander. To remove baseline drift from the ECG signals, we apply a high-pass Butterworth filter to each signal in the tensor. Baseline drift is a low-frequency noise that slowly shifts the baseline of the ECG over time, often caused by breathing or movement. The filter removes these slow trends by blocking frequencies below a certain cut-off (in our case, 0.5 Hz), while keeping the rest of the signal intact. [31]

We design a second-order Butterworth filter with a cut-off of 0.5 Hz, which is a common choice for eliminating baseline wander. The filtering is done using the filtfilt function from SciPy [33], which applies the filter forward and backwards, ensuring no phase shift is introduced and preserving the shape of the waveform. This is applied to each signal individually, meaning each lead for each ECG sample, so that the result is a clean signal with the baseline centred more stably around zero. [34]

Secondly, we remove power line interference by applying a notch filter at 50 Hz, which is the typical frequency of electrical noise. The notch filter is designed to specifically target and suppress this frequency, while leaving the rest of the signal unaffected. This helps to clean up the signal without distorting the actual rhythms. [35]

Lastly, we denoise the high-frequency noise using wavelet denoising. This technique breaks down the ECG signal into different frequency components using a so-called wavelet transform.

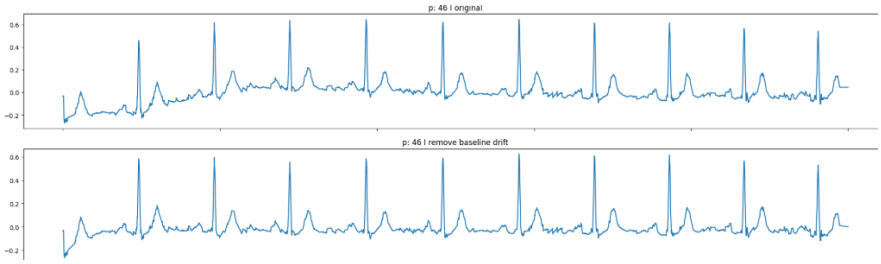


**Figure 4.3:** Baseline wander filter.

After this, it identifies which parts of the signal are likely to be noise. These are typically the small, rapid fluctuations, which are removed by a thresholding step. Once this is done, the signal is reconstructed from the remaining components. Again, this method effectively keeps the important features of the ECG while filtering out the unwanted high-frequency noise. [36]

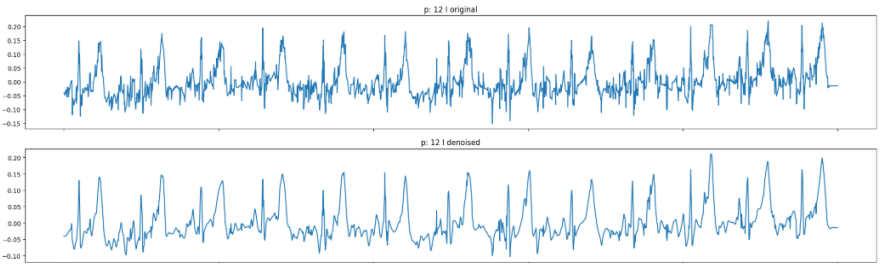


**Figure 4.4:** wavelet denoising and notch filter.

As mentioned, we experimented thoroughly with different preprocessing steps, exploring which techniques to use, which combinations worked best, and which to avoid. However, it quickly became clear that denoising is essential, as we observed that denoising consistently improved the quality of the decomposition, which is why we applied it in every case. That said, we did carefully experiment with the degree of denoising, especially for our prediction task, where overly aggressive denoising could risk removing subtle but important patterns.

### 4.6.3 Decimation

The next step is decimation, which is a specific form of downsampling designed to reduce data size while preserving the essential structure of the signal. In our dataset, the sampling frequency is 500 Hz, meaning each 10-second ECG recording contains 5000 data points. This is a substantial amount of information, much more than typically necessary to represent a signal. Having this many data points not only increases computational cost but also complicates decomposition, as the model must process a lot of redundant information. Therefore, we reduce the number of samples per signal. [37]

Suppose we want to bring this down to 1000 values. A naïve approach would be to take only the first 1000 samples, but this would cut off a large part of the signal. Another common method is to take every fifth sample, but this risks introducing aliasing, where high-frequency components are misrepresented as lower frequencies, distorting the signal.

The most appropriate approach for ECG signals is decimation, which performs downsampling in a more informed way. Before reducing the number of samples, a low-pass filter is applied to the signal. This filter removes high-frequency components that cannot be preserved when the sampling rate is lowered. The cut-off frequency is determined according to the Nyquist–Shannon sampling theorem [38], which states that the sampling rate must be at least twice the highest frequency component in the signal to avoid aliasing. After this filtering step, we can safely drop samples by taking every n-th sample without losing meaningful signal structure.

This method preserves the key morphology and rhythm of the ECG while reducing the amount of data, making the decomposition both more efficient and more robust.

This is another preprocessing step that we included in every pipeline, as downsampling is known to be very useful. The only risk is downsampling too much, which could result in the loss of important signal structure, especially for our prediction task. That is why we experimented with different downsampling factors.

For classification, we concluded that a factor of 10 was acceptable. When inspecting the signals, we noticed that some very minor high-frequency details were removed, but no essential patterns needed for classification were lost.

For prediction, we had to be more careful. We couldn't rely on just visualising the signals, since the hints of AF in SR rhythms are too subtle to detect with the naked eye. To determine a suitable downsampling factor, we analysed decomposition errors and correlation coefficients. Based on these results, we found that a factor of 4 provided a good balance.

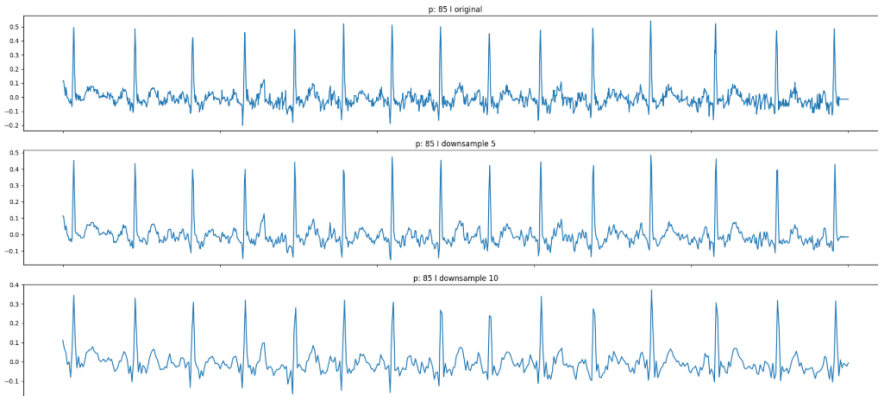


**Figure 4.5:** A decimation with factor 5 and 10.

### 4.6.4 Others

We looked further into some other preprocessing techniques, which is explained in this subsection. However, these did not make the final pipeline, which is why we don't go too much into detail.

The first one is standardisation, which gives the signal a mean of 0 (hence the signal fluctuates around 0) and a standard deviation of 1. The reason we didn't use it, is that this again, in practice puts different signals in different ranges.
Standardisation only worked after per signal normalisation, but in this case, standardisation added no value any more.

$$z = \frac{x - \mu}{\sigma}$$

The next thing we tried was per-signal max-absolute normalisation.
Earlier, we established that per-sample normalisation was useful for bringing all signals into a common range of [0,1], making them easier to compare and process. However, one limitation of this approach is that it doesn't centre the signals around zero, meaning the signals remain entirely non-negative. In our case, this raised the question: could decomposition perform better if the signals fluctuated around zero, meaning their baseline sits at zero rather than being entirely above it?

Simply shifting normalised signals to be zero-centred would again cause inconsistencies in the range between signals, defeating the purpose of normalisation. Therefore, we needed a method that not only centres signals around zero but also scales them uniformly, so each signal remains within the same bounds.

This exact combination is not straightforward to achieve. Usually, you must choose between zero-centring or range-scaling. However, we found a compromise: per-signal max-absolute normalisation.

This technique processes each signal (i.e., each lead in each sample) separately by dividing it by its maximum absolute value. The idea is to scale the signal based on its largest deviation from zero, which theoretically should bring it into the range [-1,1]. However, in practice, this doesn't happen cleanly, unless the signal is perfectly symmetric. Either the minimum or the maximum value will dominate, meaning the resulting range doesn't span the full [-1,1]. Instead, signals end up in skewed ranges like [-0.7,1] or [-1,0.5].

This subtle shift already breaks the assumption of uniform scaling across signals. So while max-absolute normalisation brings the signals closer to being centred and scaled, it still introduces enough variability in the range to cause unwanted side effects.

### Final pipeline

In the end, it quickly became clear which preprocessing steps were necessary and which weren't. The final pipeline ended up being first denoise, applying all discussed denoising techniques, then min-max normalisation per signal, and lastly a decimation. The other discussed techniques didn't make any difference, or made it significantly worse.

## 4.7 Decomposition

### 4.7.1 Decomposition choice

Initially, we explored Block Term Decomposition (BTD) [13] as a possible method for decomposing our tensor. BTD factorises the input tensor into higher-rank blocks, which in theory capture complex internal relationships between the modes of the data more effectively than simpler approaches such as CP or Tucker decomposition. However, despite this potential, BTD is not widely used in practice, primarily due to its complexity.

In contrast to CP or Tucker, BTD requires more advanced optimisation techniques. Common approaches include Gauss-Newton methods or block variants of Alternating Least Squares (ALS) [13, 39], but there is currently no widely accepted standard. As a result, popular Python libraries such as Tensorly or SciPy do not offer support for BTD. These libraries prioritise algorithms that are well understood, computationally stable, and broadly supported, which BTD has not yet achieved.

There are some public BTD implementations available in MATLAB, but these are also not yet fully developed or optimised. We considered using MATLAB, but doing so would mean carrying out most of the remaining machine learning pipeline in MATLAB as well, in order to maintain consistency. While this is possible, it does not align with the aim of this case study.

Our goal is not to improve or invent new algorithms, but rather to investigate how existing methods can best be applied and combined within a specific use case. Employing BTD would shift our focus towards research on the decomposition method itself, which is already an active area of investigation [13, 39]. Instead, we chose to work in Python, where modern machine learning tools are readily available, allowing us to concentrate on integration and evaluation, rather than algorithm development.

In summary, we did not use BTD because it is not mature enough in Python, and because using it would turn our study from an applied exploration into algorithmic research, which is beyond the intended scope.

One of the more widely used tensor decomposition methods is CANDECOMP/PARAFAC (CP) decomposition [12]. Like BTD, its goal is to factorise a tensor into components that capture meaningful structure. However, while BTD decomposes the tensor into blocks of higher rank,

CP restricts the decomposition to a sum of rank 1 tensors. This makes CP conceptually simpler and computationally more efficient.

Although this constraint may limit its ability to model complex inter-mode relationships compared to BTD, CP is significantly easier to optimise. It benefits from well-established algorithms and is supported in Python through libraries such as Tensorly, which offer stable and efficient implementations.

This practicality is the main reason we chose to use CP. While it may not capture structure as richly as BTD could, it still allows us to explore the essential principles of tensor decomposition in a robust and reproducible way.

Furthermore, because the underlying ideas of CP and BTD are related, both aiming to extract meaningful low-rank structure from tensors, our results and methodology could potentially be adapted to BTD in the future. Once BTD matures and a widely accepted, optimised standard is available in Python, much of our pipeline could be transferred or extended with minimal modification.

### 4.7.2 CP Decomposition

To perform CPD, we used the decomposition.parafac function from Tensorly v0.8.1 in Python [40].

The input in this function is our preprocessed tensor, so it used to be $(6428, 5000, 12)$, but after sampling and downsampling (decimation), this usually (depending on the exact factors and ratio) becomes something like $(1000, 500, 12)$. The function takes in a lot of arguments, which we all tested.

#### Rank

The most important parameter is the rank, which determines the number of latent factors the decomposition produces, hence the number of rank-one tensors or patterns the algorithm will attempt to extract. Determining a suitable value required experimentation.

We focused mainly on ranks between 4 and 12. Values below 4 consistently failed to capture sufficient structure, while 12 was the upper limit our local machine's memory could handle. We also tested higher ranks using a Kaggle notebook with increased RAM and GPU support, but observed no noticeable improvement beyond rank 12. In most cases, ranks above 8 yielded diminishing returns, which confirmed that continuing on our own hardware was feasible.

#### Iterations and tolerance

Two other important parameters are $'n\_iter\_max'$ (max iterations) and $'tol'$ (tolerance). The $'n\_iter\_max'$ parameter sets the maximum number of iterations the algorithm will perform. The tolerance parameter defines the minimum required decrease in reconstruction error between iterations. If the improvement falls below this threshold, the algorithm is considered to have converged and will stop early. If the error never decreases by less than $'tol'$, the algorithm will simply run for the full number of iterations defined by $'n\_iter\_max'$.

Using too few iterations can lead to underfitting, meaning the decomposition might not capture enough of the underlying structure. On the other hand, using too many iterations could result in overfitting, where the model starts to capture noise instead of meaningful patterns, potentially harming generalisation. However, in our case, overfitting seemed less of a concern, as the noise had already been largely filtered out. Moreover, once the reconstruction error starts to plateau, further iterations are unlikely to capture much more, including noise.

We experimented with various values for the tolerance. Other studies often use tolerances between $1e-4$ and $1e-8$. In our tests, the decomposition never reached a decrease smaller than $1e-5$ within the first 50 iterations. Typically, the error dropped below $1e-4$ after about

10 iterations. Although most of the improvement occurred within the first 10 iterations, we noticed that many papers accept smaller improvements and allow more iterations [41]. Based on this, we chose a tolerance of $1e-5$ and a maximum of 20 iterations, which provided a good balance between efficiency and performance.

### Initialisation

Another important parameter is *init*, which determines how the initial factor matrices are chosen before the decomposition process begins. There are a few options: $'random'$, $'svd'$, or passing a custom CPTensor. Using $'svd'$ means the factor matrices will be initialised using singular value decomposition, which often leads to faster and more stable convergence. In our case, we chose $'svd'$ as it gave more reliable results compared to random initialisation.

### Normalise factors

The next parameter in the CP decomposition is $'normalize\_factors'$. When this is set to True, each factor matrix is normalised so that the columns have unit norm. This means the strength of each component is fully captured by the weights vector. When it is set to False, the weights are simply all ones, and the scale is instead kept inside the factor matrices.

In our case, we noticed that enabling or disabling this parameter did not affect anything in practice. The reconstruction error stayed exactly the same, and the features we extracted from the factors gave the same correlation scores. This makes sense, since normalising or not only shifts the scaling between the weights and the factor matrices, but does not change the overall structure or approximation of the tensor. Also, since correlation is scale-invariant, the absolute scale of the factors does not matter for the classification task.

So technically, it does not make a difference whether this is enabled or not. However, we chose to enable it, simply because it is good practice. It makes the weights interpretable, as they reflect the overall strength of each component. It also makes the factor matrices easier to work with later on, especially if we want to compare different decompositions or do further analysis.

### Orthogonalise

The orthogonalise parameter orthogonalises the factor matrices during decomposition when set to True. This means the columns of the factor matrices will be forced to be orthogonal to each other, so that every component captures something completely different and independent from the others.

In theory, this can be useful if we want to guarantee that there is no overlap between components. However in practice, it can actually limit the decomposition, because it restricts the space in which the algorithm can search for a good solution. Forcing them to be orthogonal might result in a worse reconstruction.

In our experiments, enabling this parameter led to slightly worse results in both reconstruction error and classification performance. This suggests that the natural structure in our data benefits from allowing some overlap between components. For that reason, we left orthogonalise disabled.

### Regularisation

The next parameter in the CP decomposition is $'l2_reg'$. This stands for L2 regularisation, and it adds a penalty term to the loss function during decomposition that discourages large values in the factor matrices. Specifically, it adds the squared L2 norm (i.e., sum of squares) of all factor matrix entries, multiplied by the value of $l2_reg$, to the reconstruction error. The idea is to prevent overfitting by encouraging smaller and more stable values in the factors. [42]

By default, this parameter is set to None, meaning no regularisation is applied. In many cases, especially when the tensor is not too noisy or the rank is well-chosen, no regularisation is used. However, when the data is noisy, high-dimensional, or the rank is relatively high compared to the tensor size, enabling L2 regularisation can help avoid overfitting and improve generalisation.

In our case, since we're working on ECG classification, a task where the signal is relatively clean but the features can be subtle and high-dimensional, we experimented with different values for $l2_reg$. We found that adding a small amount of regularisation (e.g., $l2_reg = 1e-3$) improved classification performance. It also led to more stable factor matrices between runs, which was useful for interpretability.

Setting $l2_reg$ too high quickly improved the correlation, but degraded the quality of the classification. The model became too constrained and failed to capture important structure in the ECG signals, leading to higher reconstruction error and worse classification results. This confirms that while L2 regularisation can be beneficial, it should be used cautiously, and usually with small values.

Overall, we kept $l2_reg$ enabled with a low value, as it gave a slight improvement in robustness and interpretability, without noticeably hurting performance. It also helped reduce variance across decompositions, which is useful when comparing or averaging factors across subjects or time windows.
We further discussed the tuning of this parameter in Chapter 6.

**Linesearch**

Lastly, we discuss the $'linesearch'$ parameter. When this is enabled, a line search step is added to the ALS (Alternating Least Squares) optimisation after each update of the factor matrices. The idea behind this is to improve convergence by adaptively choosing a step size that minimises the reconstruction error along the update direction. [43]

By default, this parameter is set to False. In most typical use cases, especially when the data is not extremely ill-conditioned or when the decomposition rank is moderate, ALS without line search already converges well and reliably. That's why many implementations leave this off unless specifically needed.

The line search adds a small computational overhead, since it needs to evaluate the objective function at different step sizes. However, in theory it can help avoid some of the oscillations or slow convergence that can happen in standard ALS, especially in cases where the tensor is large, sparse, or difficult to decompose due to its structure.

In our experiments, enabling line search did not change the final reconstruction error or classification accuracy significantly. Convergence was sometimes slightly faster in terms of iterations, but the overall runtime was actually a bit longer due to the overhead of the line search itself. So practically, it made no real difference for our task.

That being said, it's still useful to know that this option exists. If we were dealing with noisier or larger tensors, or if ALS started to diverge or get stuck, enabling line search could help. But in our ECG classification setting, where the tensors were well-behaved and the rank was moderate, ALS converged reliably without needing any step size tuning.

So we left $'linesearch'$ disabled. It added complexity without clear benefits, and the results, both in terms of decomposition quality and correlation coefficients, stayed the same.

### 4.7.3 Visualisation of CP Decomposition Components

To get an intuitive view of what the CP decomposition has learned, we visualise the latent factors for each component. Since we decompose into R (Rank) components and our tensor has 3 modes, being samples, time (Hertz), and leads, this results in a matrix with R rows and

3 columns. where Rank is 4. Each 'cell' in this matrix shows a vector: it is the $r$-th column of the factor matrix for that mode, corresponding to component $r$. Each row represents one component across all three modes, and each column shows how that component is expressed along one specific dimension. See Figure 4.6 for an example, where Rank is 4, and each class has 100 samples.

This is also the part where CP decomposition becomes more interpretable than many black-box models like neural networks. For example, you can look at a single component and see in which time regions this pattern is active, and in which leads it appears most. This allows for more insight into what kind of ECG pattern that component is capturing, when it happens and where it shows up in the 12-lead signal. In theory, this opens the door to clinically meaningful interpretation.

However in this work, we found that the interpretability remains limited. Especially the time factor often contains high-frequency noise or fluctuations that are hard to interpret directly. Still, the fact that such a structure exists and can be visualised at all is already a promising aspect. It shows that interpretability is possible by using decomposition, even if it is not fully realised here yet.

For the sample mode, we also use colour to differentiate the binary labels, one colour for each class. This gives a rough first impression of whether a certain component might help to separate the two classes. In most cases, this separation is not clearly visible just by eye. Therefore, we turned this into a formal metric using correlation. described in the next section on feature selection. So while the visualisation doesn't immediately reveal strong class differences, we will show that these differences do exist once properly quantified.
It is important to remember that the decomposition is an unsupervised learning technique, meaning it does not use the labels of the samples during the decomposition itself. The only thing we do is keep track of the labels of the samples before decomposition, and reattach them to the sample mode afterwards. This allows us to evaluate how well the decomposition has implicitly separated the two classes, without ever having access to the label information during training.

This idea is the core of the whole technique: if certain patterns in the data, captured as components in the decomposition, appear more strongly in one class than the other, then the factor values for the sample mode will reflect this. These factor values can then be used as features for a regression model to predict the class. In Figure 4.6, we can already see early signs of this. For example, in Factor 0, the red (AF) samples tend to have higher values, while in Factor 3, the blue (SR) samples dominate. This visual separation is not always clear-cut, but it shows that the decomposition starts to uncover class-relevant structure, and that's exactly what we built on in the next step.

## 4.8 Feature selection

After the CP decomposition, we obtain a set of latent factors. To reiterate for our case, these consist of one factor matrix per mode, with the sample mode factor matrix containing the representations for each individual ECG recording. Each column in this matrix corresponds to one component, and each row to one sample. In other words, this matrix describes how much each pattern (latent factor) is found in each sample. These components serve as features for our classification model.

However, not all components are equally useful. Some might capture patterns unrelated to the labels, or mostly reflect noise. Therefore, we apply a feature selection step to determine which of the CP components separate the labels the most, and hence are most informative for classification. So, the goal is to rank or filter these latent factors based on how well they separate the binary labels.

To do this, we use correlation-based metrics. Previously, we already used Pearson correlation

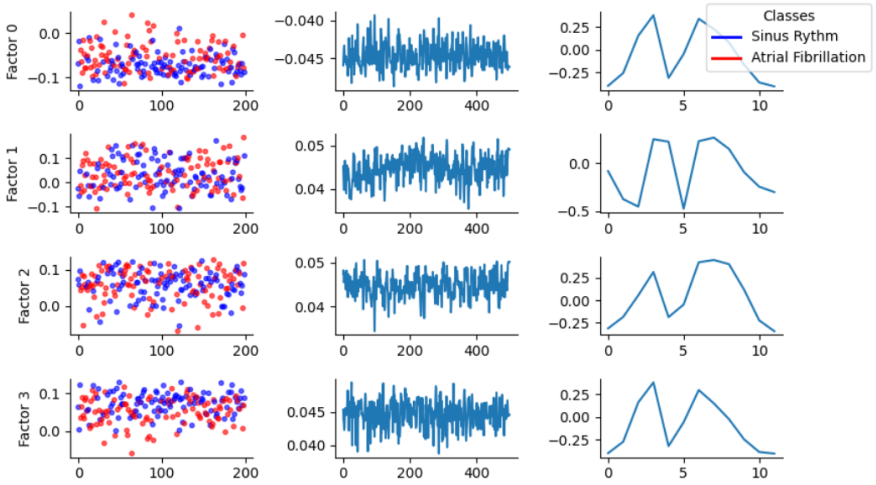


**Figure 4.6:** A visualisation of a decomposition with separation of labels. [29]

as a way to evaluate the quality of the preprocessing and decomposition. In this section, we formally explain what these correlation coefficients mean, how they work, and how they help in selecting features. We focus on two types: Pearson and Spearman correlation.

### 4.8.1 Pearson Correlation

The Pearson correlation coefficient $r$ between two variables $x$ and $y$ is defined as: [44]

$$r = \frac{\text{Cov}(x, y)}{\sigma_x \sigma_y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{4.1}$$

Here, $\bar{x}$ and $\bar{y}$ are the sample means of $x$ and $y$, and $\sigma_x$, $\sigma_y$ are the corresponding standard deviations.

We already mentioned Pearson before to evaluate how well the CP decomposition works, but we now define it more formally and explain how to interpret it. Intuitively, Pearson measures whether two variables tend to increase or decrease together in a linear way. In our case, $x$ refers to the values for one CP component across all samples (i.e., one column of the sample factor matrix), and $y$ refers to the binary labels of those samples.

Since our labels are binary (0 or 1), the correlation measures whether higher values in this factor are associated with class 1 or class 0. So, a positive Pearson correlation means that as the component value increases, the probability of class 1 increases. A negative correlation means that higher values in this factor are associated with class 0. A correlation close to 0 means there is no linear relationship.

This gives us an idea of whether a latent factor might help separate the classes, which is why we use Pearson. Not only to evaluate the quality of the decomposition, but also to assess the quality of preprocessing and to select relevant features for classification.

### 4.8.2 Spearman Correlation

The Spearman correlation coefficient $\rho$ is defined similarly to Pearson, but instead of working with raw values, it computes the Pearson correlation between the **ranked** values of the variables: [45]

$$\rho = \frac{\sum_{i=1}^{n}(R(x_i) - \bar{R}_x)(R(y_i) - \bar{R}_y)}{\sqrt{\sum_{i=1}^{n}(R(x_i) - \bar{R}_x)^2}\sqrt{\sum_{i=1}^{n}(R(y_i) - \bar{R}_y)^2}} \tag{4.2}$$

Here, $R(x_i)$ and $R(y_i)$ are the ranks of $x_i$ and $y_i$ in their respective vectors, and $\bar{R}_x$, $\bar{R}_y$ are the mean ranks.

The general intuition is the same as with Pearson: we want to know if a latent factor helps separate the two classes. The difference is that Spearman does not assume a linear relationship, but a monotonic one. It checks whether higher values of one variable tend to correspond to higher (or lower) values of the other, regardless of the exact shape of the relationship.

In theory, Spearman is useful to detect relationships that are not strictly linear but still monotonic. So we include it to double-check whether Pearson might miss something due to its linear assumption. However, in practice, we never observed a case where Spearman revealed something that Pearson did not. This suggests that the relationships between latent factor values and labels in our data are mostly linear. This is what we would expect, as the x-axis of our sample matrix is a discrete sample ID, not a continuous value.

### 4.8.3 Component selection

After performing extensive testing of the CP decomposition across various configurations, and manually inspecting the resulting Pearson correlations between the sample-mode latent factors and the binary labels, we consistently observed a small subset of components achieving a correlation above 0.1. The highest values were typically slightly above 0.2, while others tended to fall below 0.1. Based on this repeated observation, we empirically determined a Pearson correlation threshold of 0.1. [46]

This threshold was chosen as a compromise: high enough to ensure that selected components show at least a weak linear relationship with the labels, but low enough to retain multiple relevant factors. While 0.1 may seem low, it is statistically significant and meaningful in the context of noisy biomedical signals and unsupervised tensor decompositions. We therefore automated the feature selection process to retain only those components whose sample-mode factors have a Pearson correlation above 0.1, to be used as input features for the classification model. [46]

## 4.9 Feature preprocessing

Now that the CP decomposition has been performed, we obtain three factor matrices: one for each mode, samples, time (Hz), and leads. These matrices have shapes samples $samples \times R$, $time \times R$, and $leads \times R$, where $R$ is the number of components (or rank). Each column in these matrices corresponds to a latent factor, and the values indicate how strongly each sample, time point, or lead is associated with that factor.

However, from this point on, we only make use of the sample factor matrix. Again, this matrix contains, for each ECG recording (i.e., each sample), a set of $R$ values that describe how strongly that sample is associated with each of the $R$ patterns found during decomposition.

These values act as a compressed representation or feature vector for each sample, essentially summarising the ECG in terms of the discovered latent factors. Since our goal is to classify the samples (e.g. AF or SR), we only need to know how each sample relates to the patterns, not when or where those patterns occur within the signal. The time and lead factor matrices are important for interpretability and analysis of the decomposition itself, but they are not used as input to the classifier.

So, moving forward, the sample factor matrix serves as the feature matrix for classification and feature selection: each row is a sample, each column is a latent feature.

Normally, before applying any machine learning classifier, the preprocessing of the input features is a crucial step that requires careful consideration. In our case, however, the CP decomposition already acts as a significant preprocessing step, transforming raw ECGs into a lower-dimensional latent space. As a result, there is limited room for additional preprocessing.

### 4.9.1 Normalisation

Still, we observed that the resulting latent factor values, used as features for classification, can vary considerably in scale, typically within a range of approximately $\pm 10$, though this can differ per factor. Since many classification models are sensitive to feature scale, each feature must contribute equally to the decision boundary. For this reason, we apply a normalisation step using the `MinMaxScaler` from the `scikit-learn` library [47]. This scaler independently rescales each feature (i.e., each column in the sample factor matrix) to lie within the range $[0, 1]$, based on the minimum and maximum values observed for that feature across all samples.

### 4.9.2 Polynomial Features and PCA

In our experiments with linear classification models such as logistic regression, we explored using both **Polynomial Features** and **Principal Component Analysis (PCA)** as preprocessing steps. For this we used the `decomposition.PCA` and `preprocessing.PolynomialFeatures` functions from the `scikit-learn` library [47]

The idea behind polynomial features is to generate additional features by combining the original ones in non-linear ways. For instance, if we start with two features, $x_1$ and $x_2$, and apply a degree-2 polynomial transformation, we obtain a new feature set: $[x_1,\ x_2,\ x_1^2,\ x_1x_2,\ x_2^2]$ This allows a linear model to capture more complex non-linear patterns and curved decision boundaries, which would otherwise be impossible using only the original features.

However, this generates an exponential number of features. That is why we also apply PCA, a dimensionality reduction technique. It transforms the data into a new set of uncorrelated variables, called *principal components*, which are ordered by the amount of variance they capture in the original data. The main goals of PCA are to reduce noise, eliminate redundancy, and potentially improve generalisation, especially when dealing with high-dimensional feature spaces.

However, after applying both techniques in several tests, we observed little to no improvement in model performance. The polynomial transformation did not significantly improve accuracy, likely because the relationships between features were already mostly linear.

As a result, and to keep our models and pipelines simpler and more efficient, we generally choose to leave out both polynomial feature expansion and PCA in our final workflows.

## 4.10 Classification models

After dividing these features into a train and test set, like explained in subsection 4.5.2, we can start training a classification model. We trained several classification models to classify ECG samples, with the main ones being Logistic Regression, Random Forest, and XGBoost. Each of these models is briefly explained in the following subsections. The final model choice was based on empirical performance, which is discussed in more detail in the Results and Evaluation section. The parameters are handled using a grid search, which is explained further in results and evaluation, Chapter 6.

### 4.10.1 Logistic Regression

We use `LogisticRegression` from `scikit-learn` [47] to classify the samples based on their latent features. This model outputs a probability score between 0 and 1, representing the likelihood of the sample belonging to class 1. A threshold of 0.5 is used to convert this into

a binary decision. Logistic regression models a linear decision boundary, which makes it a natural first choice for classification when the data is expected to be (at least partially) linearly separable. [15]

The main parameters include:

- `penalty`: Specifies the regularisation used. Options include `'l2'` (default), which helps prevent overfitting.
- `C`: Inverse of regularisation strength. Smaller values imply stronger regularisation.
- `solver`: Algorithm used for optimisation, such as `'lbfgs'`, `'saga'`, or `'liblinear'`.
- `max_iter`: Maximum number of iterations taken for the solver to converge.

Logistic regression is well-suited to our use case because it provides a simple yet effective baseline. If strong performance is achieved with this model, it confirms that the decomposition has uncovered a meaningful and separable structure in the data.

### 4.10.2 Random Forest

Random Forest is an ensemble method that builds multiple decision trees [48] and aggregates their predictions, usually through majority voting. This approach reduces the risk of overfitting compared to a single decision tree and improves overall robustness. [16]

In our case, although we are not dealing with high-dimensional data, Random Forest remains a strong candidate because of its ability to model non-linear relationships and interactions between features, without requiring any special scaling or preprocessing. This flexibility makes it well-suited to the latent factor features produced by our tensor decomposition.

We use the implementation from `scikit-learn` [47], with the following key parameters:

`n_estimators`: the number of trees in the forest.

`max_depth`: limits the depth of the trees to prevent overfitting.

`min_samples_split`, `min_samples_leaf`: define the minimum number of samples required to split an internal node or be a leaf.

`max_features`: how many features to consider when looking for the best split.

### 4.10.3 XGBoost

XGBoost, short for Extreme Gradient Boosting, is a powerful ensemble learning method based on decision trees. Unlike Random Forest, which builds trees independently, XGBoost builds them sequentially, where each new tree focuses on correcting the errors made by the previous ones. This boosting strategy often leads to improved performance. [17]

The main parameters we considered include:

- `n_estimators` – the number of boosting rounds or trees,
- `learning_rate` – controls how much each tree contributes to the final prediction,
- `max_depth` – the maximum depth of each tree,
- `subsample` – the fraction of training samples used for each tree to reduce overfitting,
- `colsample_bytree` – the fraction of features used per tree.

XGBoost can be a strong choice for our task because of its ability to model complex, non-linear relationships between features and labels, while also including built-in regularisation to prevent overfitting. To implement this, we used the XGBoost Python Package. [17]

## 4.11 Model Evaluation

To evaluate our models, we use two approaches: **cross-validation** and a **held-out test set**. Cross-validation is generally preferred, as it avoids bias introduced by a specific train-test split. With cross-validation, we evaluate the model based on the average accuracy, precision, recall, confusion matrix, and F1-score across all folds.

However, a limitation of cross-validation is that the training and validation splits always follow the same distribution. This makes it unsuitable when we want to train on a balanced dataset while testing on a naturally imbalanced one. In such cases, we evaluate the model on a separate test set.

This natural test set is used to simulate real-world conditions by preserving the natural class distribution. We apply the trained model to the test set and evaluate the predictions using a confusion matrix, along with class-specific precision, recall, and F1-score.

## 4.12 Experimental Setup

This work was conducted using Python v3.10.8, primarily through Jupyter Notebooks (.ipynb) for experimentation and exploratory analysis. Additional functionality and reusable components were implemented in Python scripts (.py). All development took place in Visual Studio Code v1.100.0. A complete overview of the Python libraries used is provided in the Implementation section.

All experiments were conducted on a local machine running Windows 11, equipped with an 11th Gen Intel® Core™ i5-1135G7 CPU at 2.40 GHz and 16 GB of RAM. No dedicated GPU was used for this work. The computational resources were sufficient for all tasks, including tensor decompositions, model training, and evaluation.

# Chapter 5

# Implementation

In this chapter, we explain all the code that is relevant to the work, following the same general structure as outlined in the methodology chapter. For each part of the methodology, we show and explain the corresponding implementation, providing the necessary code and highlighting how it connects to the described approach. All libraries used during implementation are mentioned and discussed where relevant.

## 5.1 Libraries

The following Python libraries were used throughout the project for data manipulation, machine learning, decomposition, signal processing, and visualisation:

- `tensorly` 0.8.1 [40]
- `scikit-learn` 1.5.2 [47]
- `scipy` 1.13.0 [33]
- `PyWavelets` 1.7.0 [49]
- `numpy` 1.26.4 [50]
- `pandas` 2.2.3 [51]
- `seaborn` 0.13.2 [52]
- `matplotlib` 3.8.4 [53]
- `xgboost` 3.0.0 [17]
- `wfdb` 4.3.0 [54]

## 5.2 Dataset loading and labelling

As a first step, we define the function `get_Y` to load the metadata file from the PTB-XL dataset and enrich it with two additional label columns. This entire process is implemented using the Pandas library [51], by loading the CSV file into a Pandas DataFrame for efficient manipulation and analysis. The first column, `rhythm`, is used for the classification task. It is derived from the `scp_codes` column, which contains a dictionary of diagnostic labels per ECG sample. The extraction is done using the helper function `extract_rhythm`, which labels the entry as `SR` if the dictionary contains the key `SR`, as `AF` if it contains `AFIB`, and `VA` otherwise. This step isolates the most relevant rhythm indicators for our analysis.

```
def extract_rhythm(code_dict):
    if 'AFIB' in code_dict:
        return 'AF'
    elif 'SR' in code_dict:
        return 'SR'
    else:
        return 'OTHERS'
def get_Y(path='data/ptbxl/'):
    Y = pd.read_csv(path+'ptbxl_database.csv')
    Y = Y.reset_index().set_index('ecg_id')
    Y = Y.rename(columns={'index': 'row_id'})
    Y.scp_codes = Y.scp_codes.apply(lambda x: ast.literal_eval(x))

    Y['rhythm'] = Y['scp_codes'].apply(extract_rhythm)
```

The second label column, `health_status`, is constructed to support the prediction task. To get this, we identify patients who had both sinus rhythm (SR) and atrial fibrillation (AF) recordings. If a sinus rhythm occurred within one month prior to an AF episode for a given patient, we label that SR recording as `unhealthy`. In contrast, patients with only SR recordings and no history of AF are labelled as `healthy`. All remaining records are marked as `unknown`. These labels provide the basis for downstream tasks related to rhythm classification and patient health prediction.

In order to realise this, we now extend the `get_Y` function to construct the `health_status` label. We begin by identifying all `patient_id`s who have both SR and AF records, stored in the variable `patients_with_both`. From this subset, we extract the corresponding SR and AF samples into separate tables named `sinus_rhythms` and `af_rhythms`, respectively. These tables are then merged on `patient_id`, resulting in a combined table `merged` containing all combinations of SR and AF recordings for each of these patients.

Each row in the merged table now contains two columns for `recording_date`: one for the SR rhythm and one for the AF rhythm. We convert both columns to the appropriate `datetime` format and compute the difference in days between them, stored in a new column called `time_diff`. We then filter this table to retain only those rows where the SR recording occurred within 31 days before the AF recording. From these filtered results, we extract the `ecg_id`s corresponding to SR recordings and remove duplicates, as the same SR may appear in multiple AF pairings due to the merge.

Next, we identify the healthy samples. These are all SR records from patients who do not appear in the `patients_with_both` set. I.e., patients who have never had an AF episode in the dataset.

With both sets of `ecg_id`s (healthy and unhealthy) defined, we return to the original DataFrame and update the `health_status` column accordingly, flagging the identified rows as either `healthy` or `unhealthy`.

```
    af = Y[Y['rhythm'] == 'AF']['patient_id'].unique()
    sr = Y[Y['rhythm'] == 'SR']['patient_id'].unique()
    patients_with_both = np.intersect1d(sr, af)

    # Filter the DataFrame for sinus rhythms (SR) of these patients
    sinus_rhythms = Y[(Y['rhythm'] == 'SR') &
        (Y['patient_id'].isin(patients_with_both))]
    # Get non-sinus rhythm records for these patients
    af_rhythms = Y[(Y['rhythm'] == 'AF') &
        (Y['patient_id'].isin(patients_with_both))]
```

```
    # Merge sinus and non-sinus rhythms on patient_id
    merged = pd.merge(
        sinus_rhythms.reset_index(),
        af_rhythms[['patient_id', 'recording_date']],
        on='patient_id',
        suffixes=('_sr', '_nsr'),
    )

    # Convert recording_date columns to datetime
    merged['recording_date_sr'] =
        pd.to_datetime(merged['recording_date_sr'], errors='coerce')
    merged['recording_date_nsr'] =
        pd.to_datetime(merged['recording_date_nsr'], errors='coerce')

    merged['time_diff'] = (merged['recording_date_nsr'] -
        merged['recording_date_sr']).dt.days
    filtered_sinus_rhythms = merged[(merged['time_diff'] >= 0) &
        (merged['time_diff'] <= 31)]
    unhealthy = filtered_sinus_rhythms.drop_duplicates(subset='ecg_id')

    # get al patient ids who nevel had a non-sinus rhythm
    patients_with_only_sinus_rhythms =
        Y[~Y['patient_id'].isin(patients_with_both) & (Y['rhythm'] == 'SR')]
        ['patient_id'].unique()
    healthy_sinus_rhythms = Y[(Y['rhythm'] == 'SR') &
        (Y['patient_id'].isin(patients_with_only_sinus_rhythms))].reset_index()

    # Add a column to Y to mark if it is healthy or unhealthy
    Y['health_status'] = 'unknown'  # Default value
    Y.loc[Y.index.isin(healthy_sinus_rhythms['ecg_id']), 'health_status']
        = 'healthy'
    Y.loc[Y.index.isin(unhealthy['ecg_id']), 'health_status'] = 'unhealthy'

    return Y
```

## 5.3 Data sampling

Now that this function is defined, we can use it to obtain our DataFrame and draw a sample from it. We illustrate this for the classification task, where we select samples labelled as SR and AF from the *rhythm* column. The same procedure applies to the prediction task, where samples are selected based on the *health_status* column, distinguishing between healthy and unhealthy cases.

From this DataFrame, we require three pieces of information: the *ecg_id* (used as the table index), the *rhythm* label, and the *row_id*. The *row_id* is particularly important, as it maps each entry in the DataFrame to its corresponding sample in our ECG tensor.

In this example, we randomly select 100 samples from each class using Pandas' `sample` function, concatenate the subsets, and shuffle the combined set with `frac=1`. We then extract the *row_id*s to use as indices when accessing the corresponding ECG data from the tensor. Additionally, we store the *ecg_id*s separately so we can later reference the exact ECG records used for our classification. Finally, we also keep the corresponding class labels, which are used for decomposition visualisation.

```
df = get_Y()
```

```
df = df[['rhythm', 'row_id']]
SR_df = df[df['rhythm'] == 'SR'].sample(100, replace=False)
AF_df = df[df['rhythm'] == 'AF'].sample(100, replace=False)

df = pd.concat([SR_df, AF_df]).sample(frac=1)
indices = df.row_id.to_numpy()
np.save("output/index_output.npy", df.index.to_numpy())
labels = df['rhythm']
```

## 5.4 Load Tensor

This function, provided by PTB-XL [2,7,8], is used to load the raw ECG signals.

The function `load_raw_data` takes three arguments: the dataframe `df` (the metadata csv file), a `sampling_rate` (defaulting to 500 Hz), and a `path` to the directory containing the ECG recordings.

Depending on the desired sampling rate, it selects the appropriate filenames from the dataframe: `filename_lr` for 100 Hz or `filename_hr` for 500 Hz. It then reads each ECG recording using the `rdsamp` function from the `wfdb` library. This function returns a tuple `(signal, meta)` for each file, where we only keep the signal part.

All signals are collected into a list and then converted into a single NumPy array before being returned.

```
def load_raw_data(df, sampling_rate=500, path='data/ptbxl/'):
    if sampling_rate == 100:
        data = [wfdb.rdsamp(path+f) for f in df.filename_lr]
    else:
        data = [wfdb.rdsamp(path+f) for f in df.filename_hr]
    data = np.array([signal for signal, meta in data])
    return data
```

We can now load the tensor and apply sampling to it using our `indices` variable. This way, the resulting tensor only contains the selected samples, in preparation for the decomposition.

```
tensor = load_raw_data(df, 500)
X = tensor[indices]
```

## 5.5 ECG preprocessing

### 5.5.1 Denoising

We define a function *denoise_tensor* which applies denoising to the entire ECG tensor. The function begins by initialising a new tensor, `denoised_tensor`, using `np.zeros_like` to match the shape of the input tensor. It then iterates over all patients and leads, calling our *denoise* function on each individual signal (i.e., one lead from one patient). The denoised signal is then stored in the corresponding position in the new tensor. This approach ensures that each signal is processed independently while preserving the original tensor structure.

```
def denoise_tensor(tensor, threshold_factor=5.0):
    patients, time, leads = tensor.shape
    denoised_tensor = np.zeros_like(tensor)

    # Iterate over patients and leads
    for p in range(patients):
```

```
        for l in range(leads):
            denoised_tensor[p, :, l] = denoise(tensor[p, :, l],
            threshold_factor=threshold_factor)
    return denoised_tensor
```

We now define our *denoise* function that performs signal denoising in several steps. It takes a single ECG signal and applies a combination of baseline correction, notch filtering, and wavelet denoising.

First, the baseline drift is removed using the *remove_baseline_drift_signal* function, which applies a high-pass filter. After this, a *notch_filter* is applied to remove powerline interference, typically at 50 or 60 Hz, depending on the region.

Next, we use `PyWavelets` to decompose the signal using wavelet decomposition with the `sym4` wavelet up to level 4. This breaks the signal into multiple frequency bands. High-frequency components (the detail coefficients) are then denoised using soft thresholding. The threshold is calculated as the standard deviation of the finest-level detail coefficients multiplied by a constant *threshold_factor*, which determines the aggressiveness of the denoising.

Finally, the signal is reconstructed from the modified coefficients using inverse wavelet transform, resulting in the denoised signal.

```
def denoise(ecg_signal, fs=500, threshold_factor=5.0):
    ecg_filtered = remove_baseline_drift_signal(ecg_signal, fs=fs)
    ecg_filtered = notch_filter(ecg_signal, fs=fs)

    # Wavelet decomposition
    wavelet = 'sym4'
    level = 4
    coeffs = pywt.wavedec(ecg_filtered, wavelet, level=level)

    # Thresholding high-frequency detail coefficients
    threshold = np.std(coeffs[-1]) * threshold_factor
    coeffs_denoised = [
        pywt.threshold(c, threshold, mode='soft') if i != 0 else c
        for i, c in enumerate(coeffs)
    ]

    # Reconstruction
    denoised_signal = pywt.waverec(coeffs_denoised, wavelet)

    return denoised_signal
```

The function `remove_baseline_drift_signal` then removes baseline drift from a single ECG signal. Baseline drift is a low-frequency noise typically caused by patient movement or respiration and can distort ECG interpretation.

This function uses the `butter` and `filtfilt` functions from the `SciPy` library (`scipy.signal`) to apply a high-pass Butterworth filter.

First, it calculates the normalised cut-off frequency by dividing the chosen cut-off (default 0.5 Hz) by the Nyquist frequency, which is half the sampling frequency (`fs`). Then, it creates the filter coefficients using `butter`, and finally applies the filter to the signal using `filtfilt`. This function performs zero-phase filtering by processing the signal in both forward and reverse directions, avoiding phase distortion.

```
def remove_baseline_drift_signal(signal, cutoff=0.5, fs=500, order=2):
    nyquist = 0.5 * fs
```

```
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='high', analog=False)
    return filtfilt(b, a, signal)
```

Lastly, we look into our `notch_filter` function to remove powerline interference from an ECG signal. This kind of noise typically occurs at 50Hz (in Europe) or 60Hz (in other regions), and can strongly affect ECG analysis if not filtered out.

This function uses the `iirnotch` and `filtfilt` functions from the `SciPy` library (`scipy.signal`).

The `iirnotch` function designs a notch filter with a centre frequency of 50Hz and a quality factor `Q`, which controls the sharpness of the notch (with a default of 30.0). The filter coefficients are then used by `filtfilt` to apply zero-phase filtering, ensuring that the signal is not phase-distorted.

This effectively removes narrow-band interference from the ECG signal while preserving the underlying waveform.

```
def notch_filter(signal, fs=500, freq=50.0, Q=30.0):
    b, a = iirnotch(w0=freq, Q=Q, fs=fs)
    return filtfilt(b, a, signal)
```

### 5.5.2 Min Max normalisation

For our `normalize_per_ecg_min_max` function, we follow the same approach as in `denoise_tensor`, where we iterate over all individual ECG signals in the tensor. This time, however, we apply min-max normalisation rather than denoising.

For each signal, we compute its minimum and maximum values using NumPy's `min` and `max` functions. We then apply the standard min-max scaling formula, as described in the methodology chapter, to scale the signal values to a range between 0 and 1.

```
def normalize_per_ecg_min_max(ecg_tensor, range_min=-1, range_max=1):
    # Ensure the tensor is a numpy array
    ecg_tensor = np.array(ecg_tensor)

    # Initialize an array for the normalized tensor
    normalized_tensor = np.zeros_like(ecg_tensor)

    # Loop through each patient and lead
    for patient_idx in range(ecg_tensor.shape[0]):
        for lead_idx in range(ecg_tensor.shape[2]):
            signal = ecg_tensor[patient_idx, :, lead_idx]
            signal_min = np.min(signal)
            signal_max = np.max(signal)

            # Avoid division by zero if signal is flat
            if signal_max > signal_min:
                normalized_tensor[patient_idx, :, lead_idx] = (
                    (signal - signal_min) / (signal_max - signal_min)
                    * (range_max - range_min) + range_min
                )
            else:
                # If the signal is flat, set to the midpoint
                # of the desired range
                normalized_tensor[patient_idx, :, lead_idx]
                    = (range_max + range_min) / 2
```

```
return normalized_tensor
```

### 5.5.3 Decimation

Our last relevant preprocessing function is `downsample_decimate`, where we use the `decimate` function from the `scipy.signal` module to perform this operation. It applies an anti-aliasing low-pass filter before downsampling to preserve the signal quality and prevent distortion. The `zero_phase=True` argument ensures that the phase of the signal is not shifted by the filtering process.

The function loops over each sample in the ECG tensor and applies the decimation along the time axis (axis 0). Finally, the resulting list is converted back to a NumPy array.

```
def downsample_decimate(ecg_tensor, factor):
    return np.array([decimate(signal, factor, axis=0, zero_phase=True)
        for signal in ecg_tensor])
```

Now that we defined the needed functions, we can preprocess our tensor.

```
X = denoise_tensor(X)
X = normalize_per_ecg_min_max(X)
X = downsample_decimate(X, factor=10)
```

## 5.6 CP Decomposition

We now input our tensor $\mathbf{X}$ into the `parafac` function to perform CP decomposition. This is done using the parameters described in Chapter 4 (Methodology). The output consists of the factor matrices, a set of component weights, and the reconstruction error. Although these weights are typically used to scale the components, often by multiplying them back into the factor matrices, we leave out this step, as we normalise the factor matrices later. In that case, the weights become redundant.

```
(weights, factors), errors = parafac(X)
A = factors[0]
B = factors[1]
C = factors[2]
factors = (A, B, C)
```

### 5.6.1 Visualisation

To better understand the decomposition results, we write a function called `plot_factors_with_labels` [29], which takes the factors, the labels (e.g., sinus rhythm or atrial fibrillation), and the number of dimensions we want to show.

Each factor corresponds to a mode of the tensor: patients, time, and leads. The function creates a subplot grid where each row shows one component and each column shows one of the modes.

For the patient mode, we use a scatter plot. Each point is coloured based on the patient's label (blue for sinus rhythm, red for atrial fibrillation or ventricular arrhythmia). For the other modes, we use line plots to show the variation of each component over time or leads.

This way, we can visually inspect whether the extracted factors show a structure that aligns with the known labels, helping us interpret what the decomposition is capturing.

```
def plot_factors_with_labels(factors, labels, d=3):
    a = factors
```

```
    rank = a[0].shape[1]
    fig, axes = plt.subplots(rank, d, figsize=(8, int(rank * 1.2 + 1)))
    factors_name = ["Patients", "Time", "Leads"] if d == 3
        else ["Time", "Features"]

    # Normalize labels for consistent plotting (0 = blue, 1 = red)
    colors = np.array(['blue', 'red'])[labels.astype(int)]

    for ind, (factor, axs) in enumerate(zip(factors[:d], axes.T)):
        axs[-1].set_xlabel(factors_name[ind])
        for i, (f, ax) in enumerate(zip(factor.T, axs)):
            sns.despine(top=True, ax=ax)
            # Scatter points for binary labels
            if ind == 0:
                ax.scatter(range(len(f)),
                    f, c=colors, alpha=0.6, s=10, label="Labels")
            else:  # Line plot for other dimensions
                ax.plot(f)
            axes[i, 0].set_ylabel("Factor " + str(i))

labels = labels.replace('SR', 0).replace('AF', 1)
plot_factors_with_labels(factors, labels, d=3)
```

### 5.6.2 Correlation

To evaluate the relationship between the extracted patient factors and the labels, we compute the Pearson correlation coefficient for each factor. This is done by comparing each column of the patient factor matrix $A$ with the label vector. The result is a correlation score and a $p$-value for each factor, indicating how strongly that factor linearly separates the labels. These values are then printed to help assess which factors may carry relevant information.

```
correlations = np.array([pearsonr(A[:, i], labels) for i in
range(A.shape[1])])

for i, (corr, pval) in enumerate(correlations):
    print(f"Factor {i}: Pearson correlations = {corr:.4f},
        p-value = {pval:.4g}")
```

## 5.7 classification

### 5.7.1 Preprocessing and sampling

The code first binarises the class labels by assigning 0 to *'SR'* and 1 to the other labels, which is *AF*. Then, it uses MinMaxScaler from the *sklearn.preprocessing* library [47] to scale the input matrix X (in this case, the samples factor matrix A from the decomposition) to the [0,1] range. The data is then split using the custom *natural_split* function, which creates a training set with a balanced class distribution (80% of the data), and a test set with a natural class imbalance (test ratio of 4.0 between class 0 and class 1), based on the specified random state.

```
    labels = labels.apply(lambda x: 0 if x == 'SR' else 1)
    Y = labels.to_numpy()
    norm_scaler = MinMaxScaler()
    A = norm_scaler.fit_transform(X)
    X_train, y_train, X_test, y_test = natural_split(
        A, Y, rs=randomsstate, train_size=0.8, test_ratio=4.0)
```

As mentioned, the function *natural_split* creates a train-test split where the training set is balanced across classes, and the test set follows a specified class imbalance (controlled by *test_ratio*). It first shuffles the data per class, then selects equal numbers of samples per class for training. For the test set, it selects as many samples as possible while maintaining the desired ratio (e.g., 4:1 for class 0 vs class 1). The function returns the split feature and label sets for training and testing.

```
def natural_split(X, Y, rs=42, train_size=0.8, test_ratio=4.0):
    np.random.seed(rs)

    # Get indices per class
    class_0_indices = np.where(Y == 0)[0]
    class_1_indices = np.where(Y == 1)[0]

    # Shuffle
    np.random.shuffle(class_0_indices)
    np.random.shuffle(class_1_indices)

    # Total samples per class (assumed equal)
    N = min(len(class_0_indices), len(class_1_indices))

    # Determine number of train samples per class
    train_samples_per_class = int(train_size * N)

    # Remaining for test
    class_0_remaining = N - train_samples_per_class
    class_1_remaining = N - train_samples_per_class

    # Determine test samples from each class to match ratio
    max_test_1 = class_1_remaining
    max_test_0 = class_0_remaining

    # Solve for x such that class_0_test = test_ratio * class_1_test
    class_1_test = min(max_test_1, int(max_test_0 / test_ratio))
    class_0_test = int(test_ratio * class_1_test)

    # Slice indices
    class_0_train = class_0_indices[:train_samples_per_class]
    class_1_train = class_1_indices[:train_samples_per_class]

    class_0_test = class_0_indices[
        train_samples_per_class:train_samples_per_class + class_0_test]
    class_1_test = class_1_indices[
        train_samples_per_class:train_samples_per_class + class_1_test]

    # Combine indices
    train_indices = np.concatenate([class_0_train, class_1_train])
    test_indices = np.concatenate([class_0_test, class_1_test])

    # Shuffle combined sets
    np.random.shuffle(train_indices)
    np.random.shuffle(test_indices)

    # Subset data
    X_train, y_train = X[train_indices], Y[train_indices]
```

```
    X_test, y_test = X[test_indices], Y[test_indices]

    return X_train, y_train, X_test, y_test
```

If we want a balanced test set, we can just use the *train_test_split(X, Y, test_size=0.2)* function from sklearn [47].

### 5.7.2 Models

Next we train our model using our train data, which is done in the code below. This code performs a grid search with 5-fold cross-validation to find the best hyperparameters for a *LogisticRegression* model. The best parameters are then used to create and train a new model. This process is the same for *RandomForestClassifier* and *XGBClassifier*.

```
model = LogisticRegression(random_state=randomsstate)
model = GridSearchCV(model, param_grid_rf, cv=5, scoring='accuracy')
model.fit(X_train, Y_train)

print("Beste parameters:", model.best_params_)
print("Beste score:", model.best_score_)

model = LogisticRegression(random_state=randomsstate, params...)
```

### 5.7.3 Test

The following code is used to evaluate the trained model on the test set. It prints the overall accuracy, a classification report with precision, recall and F1 score for each class, and the confusion matrix.

```
y_pred = model.predict(X_test)

accuracy = accuracy_score(Y_test, y_pred)
print("Accuracy:", accuracy)
print(classification_report(Y_test, y_pred))
print(confusion_matrix(Y_test, y_pred))
```

### 5.7.4 Cross validation

This part performs 5-fold stratified cross-validation on the training set. Multiple scoring metrics are defined, and the average scores across folds are computed. This provides a more robust estimate of the model's performance.

```
scoring = {
    'precision_macro': make_scorer(precision_score, average='macro'),
    'recall_macro': make_scorer(recall_score, average='macro'),
    'f1_macro': make_scorer(f1_score, average='macro'),
    'accuracy': make_scorer(accuracy_score)
}
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=rs)
scores = cross_validate(model, X_train, Y_train, cv=cv, scoring=scoring)

print("Avg precision:", scores['test_precision_macro'].mean())
print("Avg recall:", scores['test_recall_macro'].mean())
print("Avg F1:", scores['test_f1_macro'].mean())
print("Avg accuracy:", scores['test_accuracy'].mean())
```

# Chapter 6

# Results and Evaluation

In this chapter, we present the final results concerning our model and parameter choices, as well as the overall model performance.
Most tests are performed within the classification task, as we had more data available for this, which makes the results more reliable. Additional results for the prediction task are presented at the end of this chapter.

## 6.1 Preprocessing

We begin by examining the impact of several preprocessing parameters in our final signal processing pipeline for our classification task. These include the denoising threshold, the wavelet decomposition level, and the decimation factor. In Table 6.1, we show four heatmaps, one for each decimation factor, showing the average absolute correlation (in percentage) for various combinations of denoising threshold and wavelet level. These results are further summarised in Table 6.2, which also includes the corresponding reconstruction error for the top 5 configurations, which remained very consistent.

In these experiments, all other parameters were kept constant. This includes setting the random state to 42 in both the sampling functions and the CP decomposition. The CP decomposition was performed with a fixed rank of 8 and an $\ell_2$ regularisation parameter of 2. These tests were performed on 100 random samples of each class.

Overall, we observe that variations in these preprocessing parameters have a relatively small effect on the decomposition outcome. In contrast, the choice of preprocessing techniques themselves had a significant influence. As described earlier in the methodology, alternative approaches led to almost no meaningful correlation, in contrast to this pipeline, which consistently produced interpretable and relevant components.

## 6.2 Decomposition Rank and Regularisation

We now take a closer look at the two most important parameters for the decomposition: Rank and L2 regularisation. For these experiments, we use the optimal ECG preprocessing setup described earlier. This includes level 3 wavelet denoising with a threshold of 7 and decimation by a factor of 20. All other settings, such as the sampling rate and random state, remain the same as in the preprocessing stage.

Looking at the results in Table 6.3, we observe that the average absolute correlation increases as we apply stronger L2 regularisation. In contrast, higher Rank values tend to reduce the correlation. When we examine the summary in Table 6.4, which highlights the best combinations based on correlation, we notice that the reconstruction errors are generally worse than those

**Table 6.1:** Heatmaps for average correlation values over threshold and wavelet level, per decimation factor.

**(a)** Decimation = 2

| Level \ Threshold | 3 | 5 | 7 |
|---|---|---|---|
| 3 | 16.8 | 15.4 | 14.9 |
| 4 | 15.8 | 15.7 | 14.9 |
| 5 | 15.4 | 15.0 | 17.0 |

**(b)** Decimation = 5

| Level \ Threshold | 3 | 5 | 7 |
|---|---|---|---|
| 3 | 16.7 | 15.2 | 14.7 |
| 4 | 15.1 | 16.6 | 14.6 |
| 5 | 17.9 | 17.4 | 17.2 |

**(c)** Decimation = 10

| Level \ Threshold | 3 | 5 | 7 |
|---|---|---|---|
| 3 | 14.5 | 16.7 | 16.8 |
| 4 | 16.8 | 15.8 | 17.1 |
| 5 | 16.7 | 15.8 | 14.0 |

**(d)** Decimation = 20

| Level \ Threshold | 3 | 5 | 7 |
|---|---|---|---|
| 3 | 13.9 | 13.7 | 19.3 |
| 4 | 15.0 | 17.6 | 17.2 |
| 5 | 16.5 | 16.8 | 16.0 |

**Table 6.2:** Top 5 parameter combinations by average correlation

| Threshold | Level | Decimation | Correlation | Reconstruction Error |
|---|---|---|---|---|
| 7 | 3 | 20 | 0.193 | 0.604 |
| 3 | 5 | 5 | 0.179 | 0.632 |
| 5 | 4 | 20 | 0.176 | 0.605 |
| 5 | 5 | 5 | 0.174 | 0.632 |
| 7 | 5 | 5 | 0.172 | 0.632 |

from the preprocessing experiments, where we used Rank 8 with an L2 regularisation of 2.0. However, since our main goal is to feed these decompositions into a classification task, correlation is more relevant than reconstruction error. This makes the higher-correlation settings more favourable, even if they result in slightly less accurate reconstructions.

## 6.3 Model Performance over Rank and Regularisation

Now that we have determined the optimal decomposition parameters, we move on to the classification task, where we try to find our optimal model. For this, we apply all previously found settings for preprocessing: wavelet denoising at level 3 with a threshold of 7, a decimation factor of 20.
Also, we use a balanced dataset with a total of 500 samples per class, 400 for training and 100 for testing.

We chose not to rely solely on the previously determined decomposition parameters, which favoured low rank and high regularisation. Instead, we re-evaluate a range of ranks and regularisation values across three different classification methods. This time, we evaluate the configurations based on classification metrics rather than correlation, using cross-validation. The goal is to investigate whether high average correlation truly translates to strong classification performance, and to determine if low-rank, high-regularisation settings indeed give the best results.

The second goal is to find the best classifier, comparing three classification models: *Logistic Regression*, *Random Forest*, and *XGBoost*. To determine the best configuration for each model, we used a grid search. As a result, the optimal parameters were as follows. For `RandomForestClassifier`, we used `bootstrap=True`, `max_depth=10`, `min_samples_leaf=5`,

**Table 6.3:** Correlation values over L2 Regularisation and Rank.

| | L2 Regularisation | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Rank** | 1e-6 | 1e-5 | 1e-4 | 1e-3 | 1e-2 | 1e-1 | 1.0 | 2.0 | 5.0 | 10.0 |
| 4 | 15.5 | 15.6 | 14.4 | 15.2 | 14.1 | 14.8 | 17.0 | 20.9 | 19.9 | 25.0 |
| 5 | 14.8 | 13.7 | 16.4 | 14.2 | 13.2 | 13.5 | 16.6 | 18.5 | 22.0 | 25.0 |
| 6 | 13.1 | 12.2 | 13.3 | 14.1 | 12.0 | 10.0 | 16.4 | 16.7 | 21.6 | 25.0 |
| 7 | 12.7 | 13.3 | 11.6 | 11.7 | 11.6 | 10.7 | 16.7 | 16.2 | 21.7 | 24.9 |
| 8 | 11.3 | 10.6 | 11.1 | 11.7 | 10.8 | 11.2 | 14.3 | 15.6 | 20.8 | 24.3 |
| 9 | 8.7 | 10.3 | 11.7 | 11.0 | 10.3 | 12.3 | 15.1 | 16.2 | 18.9 | 23.6 |
| 10 | 10.8 | 11.7 | 10.3 | 10.7 | 11.1 | 9.5 | 15.6 | 15.1 | 18.6 | 23.5 |
| 11 | 8.2 | 10.2 | 9.4 | 10.0 | 9.2 | 11.4 | 15.0 | 16.3 | 19.9 | 23.2 |
| 12 | 8.5 | 11.6 | 8.7 | 8.4 | 9.9 | 8.9 | 13.7 | 15.0 | 16.5 | 22.4 |

**Table 6.4:** Top 5 parameter combinations for decomposition by average correlation

| Threshold | Level | Correlation | Reconstruction Error |
|---|---|---|---|
| 10.0 | 4 | 0.250 | 0.863 |
| 10.0 | 6 | 0.250 | 0.826 |
| 10.0 | 5 | 0.250 | 0.843 |
| 10.0 | 7 | 0.249 | 0.811 |
| 10.0 | 8 | 0.243 | 0.799 |

`min_samples_split=2`, and `n_estimators=100`.
For `XGBClassifier`, the best configuration was `colsample_bytree=0.7`, `gamma=0`, `learning_rate=0.01`, `max_depth=3`, `n_estimators=50`, `reg_alpha=0`, `reg_lambda=0`, `scale_pos_weight=1`, and `subsample=0.7`. Lastly, for `LogisticRegression`, the best parameters were `C=1`, `max_iter=100`, `penalty='l2'`, and `solver='saga'`.
Upon examining the results of the grid search, we observed that varying these parameters led to very similar outcomes. Therefore, we chose to keep these parameters fixed in order to simplify the analysis and focus on those that had a more significant impact. In this case, the rank, regularisation strength, and model choice.

Whereas in earlier experiments we used a fixed random state of 42, we now repeat the classification using five different random states: 42, 26, 87, 36, 14. We then average the results. This is because the classification process is more sensitive to randomness, due to factors such as the train-test split, undersampling, shuffling, model training, and the cross-validation procedure, where we all use a random state. Note that we also still do the sampling, preprocessing and decomposition steps with these random states.

In Table 6.5, we present three heatmaps for each classifier: the top one shows accuracy, while the two below display the F1 scores for each class. Then, in Table 6.6, we list the four configurations with the highest accuracy from Table 6.5, along with their confusion matrices and precision and recall values for each class. In both tables, all results are averaged over the random states and cross-validation folds, except for the confusion matrix values, which are summed.

We can see that Logistic Regression delivers the best results. It's also clear that the low rank and high regularisation, which gave the best average correlation, do not produce the best classification outcomes. Generally, higher ranks tend to give better results, while low to moderate regularisation performs best.
Additionally, the model's predictions are consistent across both classes, with precision and recall values remaining well balanced.

Then, in Table 6.7, we present the same four optimal models as before. This time, however, they are evaluated on a test set that reflects a more natural, real-world class distribution, with 100 samples for class 0 and 25 samples for class 1. Since this test set does not support cross-validation, each model was trained and tested once per random state without performing multiple folds.

We still observe roughly the same accuracy levels; however, the confusion matrix and precision and recall scores are no longer balanced. The model now predicts class 0 more reliably than class 1.

## 6.4 prediction

We now dive into the prediction task. Most of the previous findings also apply to this, which is why we didn't repeat all steps. The main thing we were interested in was the preprocessing, as we expected a difference in optimal parameters here. With classification, we downsampled our data by a factor 20, however in this case we assumed downsampling this much would mean losing the subtle patterns we are looking for in this case, which is why we reperformed the tests for the optimal preprocessing parameters: denoising threshold, the wavelet decomposition level, and the decimation factor.

In these tests, we kept other parameters constant, learning from previous tests. We chose a Rank 8 decomposition with a regularisation of $0,1$. For sampling, we only have 42 class 1 samples. Because we want a balanced training set, and an unbalanced test set, we went for 32 training samples per class. For the test set, we went for 40 class 0 and 10 class 1 samples.

These results can be seen in Table 6.8, where we can again see 4 heatmaps, one for each decimation value, with each heatmap being the accuracy for denoising threshold over wavelet decomposition.

Subsequently, in Table 6.9, we give an overview of the top 4 configurations from Table 6.8, based on accuracy, showing the confusion matrices and precision and recall for each class.

**Table 6.5:** Heatmaps per model showing accuracy and class-specific F1 scores across regularisation and rank.

**(a)** xgb – Accuracy

**L2 Regularisation**

| Rank | 1e-05 | 1e-03 | 1e-01 | 1 | 10 |
|---|---|---|---|---|---|
| 4 | 61.1 | 61.3 | 62.1 | 60.8 | 57.2 |
| 6 | 64.1 | 63.3 | 63.2 | 60.8 | 56.9 |
| 8 | 63.0 | 63.2 | 63.8 | 61.3 | 57.3 |
| 10 | 62.7 | 63.6 | 64.2 | 63.1 | 57.4 |

**(b)** xgb – F1 Class 0

**L2 Regularisation**

| Rank | 1e-05 | 1e-03 | 1e-01 | 1 | 10 |
|---|---|---|---|---|---|
| 4 | 62.8 | 63.1 | 63.9 | 62.0 | 63.4 |
| 6 | 65.3 | 64.4 | 64.5 | 61.9 | 63.0 |
| 8 | 63.9 | 64.1 | 65.0 | 62.8 | 62.5 |
| 10 | 63.8 | 64.5 | 64.9 | 64.6 | 63.4 |

**(c)** xgb – F1 Class 1

**L2 Regularisation**

| Rank | 1e-05 | 1e-03 | 1e-01 | 1 | 10 |
|---|---|---|---|---|---|
| 4 | 59.2 | 59.2 | 60.1 | 59.3 | 48.4 |
| 6 | 62.8 | 62.1 | 61.9 | 59.6 | 48.3 |
| 8 | 62.1 | 62.3 | 62.3 | 59.6 | 50.2 |
| 10 | 61.4 | 62.7 | 63.5 | 61.5 | 49.1 |

**(d)** RandomForest – Accuracy

**L2 Regularisation**

| Rank | 1e-05 | 1e-03 | 1e-01 | 1 | 10 |
|---|---|---|---|---|---|
| 4 | 59.6 | 60.4 | 61.2 | 59.1 | 54.6 |
| 6 | 63.0 | 63.1 | 63.3 | 59.5 | 54.7 |
| 8 | 62.5 | 63.4 | 63.4 | 59.6 | 53.8 |
| 10 | 62.8 | 62.4 | 63.0 | 61.2 | 54.5 |

**(e)** RandomForest – F1 Class 0

**L2 Regularisation**

| Rank | 1e-05 | 1e-03 | 1e-01 | 1 | 10 |
|---|---|---|---|---|---|
| 4 | 60.6 | 61.4 | 62.1 | 60.0 | 56.5 |
| 6 | 63.7 | 63.9 | 64.2 | 60.3 | 56.9 |
| 8 | 63.1 | 64.1 | 64.1 | 60.3 | 55.8 |
| 10 | 63.6 | 62.5 | 63.6 | 61.9 | 57.1 |

**(f)** RandomForest – F1 Class 1

**L2 Regularisation**

| Rank | 1e-05 | 1e-03 | 1e-01 | 1 | 10 |
|---|---|---|---|---|---|
| 4 | 58.5 | 59.2 | 60.3 | 58.2 | 52.5 |
| 6 | 62.3 | 62.3 | 62.4 | 58.7 | 52.2 |
| 8 | 61.8 | 62.6 | 62.8 | 59.0 | 51.5 |
| 10 | 62.0 | 62.2 | 62.4 | 60.4 | 51.7 |

**(g)** LogisticRegression – Accuracy

**L2 Regularisation**

| Rank | 1e-05 | 1e-03 | 1e-01 | 1 | 10 |
|---|---|---|---|---|---|
| 4 | 60.9 | 60.8 | 60.7 | 60.9 | 58.4 |
| 6 | 64.4 | 64.3 | 64.4 | 60.7 | 58.3 |
| 8 | 64.2 | 64.2 | 64.4 | 60.8 | 58.3 |
| 10 | 64.5 | 64.3 | 64.2 | 62.3 | 58.1 |

**(h)** LogisticRegression – F1 Class 0

**L2 Regularisation**

| Rank | 1e-05 | 1e-03 | 1e-01 | 1 | 10 |
|---|---|---|---|---|---|
| 4 | 62.2 | 62.1 | 61.9 | 62.1 | 61.0 |
| 6 | 64.5 | 64.3 | 64.4 | 61.9 | 61.0 |
| 8 | 64.0 | 64.2 | 64.3 | 61.8 | 61.0 |
| 10 | 64.4 | 64.3 | 64.2 | 63.2 | 60.9 |

**(i)** LogisticRegression – F1 Class 1

**L2 Regularisation**

| Rank | 1e-05 | 1e-03 | 1e-01 | 1 | 10 |
|---|---|---|---|---|---|
| 4 | 59.5 | 59.5 | 59.4 | 59.6 | 55.4 |
| 6 | 64.4 | 64.3 | 64.3 | 59.4 | 55.3 |
| 8 | 64.4 | 64.3 | 64.4 | 59.6 | 55.1 |
| 10 | 64.5 | 64.2 | 64.2 | 61.5 | 54.8 |

**Table 6.6:** Top 4 Models: Confusion Matrix and Precision / Recall per Class

*LogisticRegression, Rank=10, $\lambda = 1e\text{-}05$, Accuracy=64.5%*

**Confusion Matrix**

| | **Pred 0** | **Pred 1** |
|---|---|---|
| **Actual 0** | 1611 (TP) | 889 (FN) |
| **Actual 1** | 888 (FP) | 1612 (TN) |

**Precision / Recall**

| | **Class 0** | **Class 1** |
|---|---|---|
| Precision | 64.5% | 64.4% |
| Recall | 64.5% | 64.5% |

*LogisticRegression, Rank=6, $\lambda = 1e\text{-}05$, Accuracy=64.4%*

**Confusion Matrix**

| | **Pred 0** | **Pred 1** |
|---|---|---|
| **Actual 0** | 1614 (TP) | 886 (FN) |
| **Actual 1** | 893 (FP) | 1607 (TN) |

**Precision / Recall**

| | **Class 0** | **Class 1** |
|---|---|---|
| Precision | 64.4% | 64.6% |
| Recall | 64.5% | 64.3% |

*LogisticRegression, Rank=6, $\lambda = 0.1$, Accuracy=64.4%*

**Confusion Matrix**

| | **Pred 0** | **Pred 1** |
|---|---|---|
| **Actual 0** | 1614 (TP) | 886 (FN) |
| **Actual 1** | 896 (FP) | 1604 (TN) |

**Precision / Recall**

| | **Class 0** | **Class 1** |
|---|---|---|
| Precision | 64.3% | 64.6% |
| Recall | 64.4% | 64.2% |

*LogisticRegression, Rank=8, $\lambda = 0.1$, Accuracy=64.4%*

**Confusion Matrix**

| | **Pred 0** | **Pred 1** |
|---|---|---|
| **Actual 0** | 1603 (TP) | 897 (FN) |
| **Actual 1** | 885 (FP) | 1615 (TN) |

**Precision / Recall**

| | **Class 0** | **Class 1** |
|---|---|---|
| Precision | 64.4% | 64.1% |
| Recall | 64.3% | 64.6% |

**Table 6.7:** Top 4 Models tested over a natural test set: Confusion Matrix and Precision / Recall per Class

*LogisticRegression, Rank=10, $\lambda = 1e{-}05$, Accuracy=63.5%*

**Confusion Matrix**

| | **Pred 0** | **Pred 1** |
|---|---|---|
| **Actual 0** | 311 (TP) | 189 (FN) |
| **Actual 1** | 39 (FP) | 86 (TN) |

**Precision / Recall**

| | **Class 0** | **Class 1** |
|---|---|---|
| Precision | 88.7% | 31.9% |
| Recall | 62.2% | 68.8% |

*LogisticRegression, Rank=6, $\lambda = 1e{-}05$, Accuracy=64.6%*

**Confusion Matrix**

| | **Pred 0** | **Pred 1** |
|---|---|---|
| **Actual 0** | 319 (TP) | 181 (FN) |
| **Actual 1** | 40 (FP) | 85 (TN) |

**Precision / Recall**

| | **Class 0** | **Class 1** |
|---|---|---|
| Precision | 88.8% | 32.3% |
| Recall | 63.8% | 68.0% |

*LogisticRegression, Rank=6, $\lambda = 1e{-}01$, Accuracy=64.3%*

**Confusion Matrix**

| | **Pred 0** | **Pred 1** |
|---|---|---|
| **Actual 0** | 317 (TP) | 183 (FN) |
| **Actual 1** | 40 (FP) | 85 (TN) |

**Precision / Recall**

| | **Class 0** | **Class 1** |
|---|---|---|
| Precision | 88.7% | 32.1% |
| Recall | 63.4% | 68.0% |

*LogisticRegression, Rank=8, $\lambda = 1e{-}01$, Accuracy=63.8%*

**Confusion Matrix**

| | **Pred 0** | **Pred 1** |
|---|---|---|
| **Actual 0** | 316 (TP) | 184 (FN) |
| **Actual 1** | 42 (FP) | 83 (TN) |

**Precision / Recall**

| | **Class 0** | **Class 1** |
|---|---|---|
| Precision | 88.1% | 31.7% |
| Recall | 63.2% | 66.4% |

**Table 6.8:** Heatmaps for prediction accuracy (in %) values averaged over 5 random states, per decimation factor.

**(a)** Decimation = 1

| **Level** \ **Threshold** | 3 | 5 | 7 |
|---|---|---|---|
| 3 | 53 | 58 | 55 |
| 4 | 56 | 56 | 55 |
| 5 | 56 | 56 | 55 |

**(b)** Decimation = 2

| **Level** \ **Threshold** | 3 | 5 | 7 |
|---|---|---|---|
| 3 | 56 | 56 | 56 |
| 4 | 56 | 57 | 59 |
| 5 | 56 | 58 | 59 |

**(c)** Decimation = 4

| **Level** \ **Threshold** | 3 | 5 | 7 |
|---|---|---|---|
| 3 | 56 | 56 | 55 |
| 4 | 56 | 56 | 55 |
| 5 | 55 | 56 | 55 |

**(d)** Decimation = 8

| **Level** \ **Threshold** | 3 | 5 | 7 |
|---|---|---|---|
| 3 | 56 | 57 | 57 |
| 4 | 57 | 57 | 59 |
| 5 | 57 | 57 | 57 |

**Table 6.9:** Top 4 prediction Models: Confusion Matrix and Precision / Recall per Class

*Decim=2, Level=4, $T = 7$, Accuracy=59.6%*

| **Confusion Matrix** | | | **Precision / Recall** | | |
|---|---|---|---|---|---|
| | **Pred 0** | **Pred 1** | | **Class 0** | **Class 1** |
| **Actual 0** | 119 (TP) | 81 (FN) | Precision | 85.6% | 27.3% |
| **Actual 1** | 20 (FP) | 30 (TN) | Recall | 59.5% | 60.0% |

*Decim=2, Level=5, $T = 7$, Accuracy=59.6%*

| **Confusion Matrix** | | | **Precision / Recall** | | |
|---|---|---|---|---|---|
| | **Pred 0** | **Pred 1** | | **Class 0** | **Class 1** |
| **Actual 0** | 119 (TP) | 81 (FN) | Precision | 85.6% | 27.3% |
| **Actual 1** | 20 (FP) | 30 (TN) | Recall | 59.5% | 60.0% |

*Decim=8, Level=4, $T = 7$, Accuracy=59.2%*

| **Confusion Matrix** | | | **Precision / Recall** | | |
|---|---|---|---|---|---|
| | **Pred 0** | **Pred 1** | | **Class 0** | **Class 1** |
| **Actual 0** | 117 (TP) | 83 (FN) | Precision | 86.2% | 27.1% |
| **Actual 1** | 19 (FP) | 31 (TN) | Recall | 58.5% | 62.0% |

*Decim=2, Level=5, $T = 5$, Accuracy=58.4%*

| **Confusion Matrix** | | | **Precision / Recall** | | |
|---|---|---|---|---|---|
| | **Pred 0** | **Pred 1** | | **Class 0** | **Class 1** |
| **Actual 0** | 117 (TP) | 83 (FN) | Precision | 84.8% | 26.1% |
| **Actual 1** | 21 (FP) | 29 (TN) | Recall | 58.5% | 58.0% |

# Chapter 7

# Discussion

In this chapter, we take a closer look at the results from our experiments. We begin with a brief summary of the main outcomes for both the classification and prediction tasks.
Next, we reflect on several important observations, focusing on model behaviour, parameter influence, and decomposition quality.
Finally, we compare our results with those from related work.

## 7.1 Overview of Results

Our best model for the classification task achieved an average accuracy of around 65% across multiple random seeds, with average recall and precision values being roughly similar. However, a more detailed look at the per-class metrics with a natural test set distribution revealed an imbalance in performance. The model consistently classified the SR class (class 0) much more accurately than the AF class (class 1). SR precision was around 88%, and a recall of 62%, indicating that the model was relatively confident and correct when predicting SR. In contrast, the AF class suffered from lower precision (averaging around 32%), but still good recall (around 69%), due to frequent false positives and overall weaker separation.

While these results were somewhat lower than we initially hoped for, especially when compared to some related works, the results are still promising, especially considering the SR classification.

For the prediction task, which attempted to predict the risk of future AF based on seemingly normal sinus rhythms, the best models reached an accuracy of around 59%. While modest, these results are consistently above chance level (50%), even with limited data. This supports earlier findings in related works using neural networks that signs of AF can be present in SR, and shows that our decomposition-based approach was able to extract some of this structure.

These results aren't ready yet for medical use, and we can name two main reasons for these weak results. First of all, the data, which was especially a problem for the prediction task, as there were very few SR rhythms in the dataset that occurred at most one month before an AF signal from the same patient.
A second issue could be the use of CP decomposition, compared to a complete Block Term Decomposition. A well-defined and properly implemented BTD, which isn't available yet, could capture many more relationships between the leads, potentially resulting in better separation.

## 7.2 Key Findings and Interpretation

One of the most notable observations from the classification results was the difficulty in correctly identifying AF rhythms, considering the difference in results between both classes. A likely explanation is that SR patterns tend to be more uniform, whereas AF signals are more irregular and variable across samples, making them harder to generalise.

One of the most important insights we gained during the decomposition and classification process was the influence of the L2 regularisation parameter. We found that higher values of this parameter led to better separation between classes in the latent space, as seen in stronger average correlation patterns in the factor matrices. This makes sense, as L2 regularisation reduces overfitting by penalising large weights, encouraging the model to focus more on the overall structure of the signal rather than on sample-specific noise.

However, despite improving correlation, a higher regularisation did not lead to better classification results. Interestingly, this was the only parameter for which improved correlation did not translate to higher classification accuracy — something that went against our initial expectations.

This suggests that strong regularisation helps identify common structures that distinguish the classes, but these structures may not be distinct enough from each other. In other words, the extracted patterns may highlight similar or overlapping aspects of the signal, reducing their usefulness for fine-grained classification. On the other hand, with lower regularisation, the class separation in terms of correlation may be slightly worse, but the resulting latent patterns are more diverse and complementary. This increased variety provides richer and more discriminative information for the classifier, ultimately improving classification performance.

Alongside the increase in regularisation, we also observed a rise in reconstruction error, despite the improved correlation. This further supports the idea that the latent factors may not have improved as much as the correlation values initially suggested, indicating that higher correlation does not necessarily translate to more informative or higher-quality components.

Throughout this work, it became clear that the decomposition was the most decisive step in the entire pipeline. Without a meaningful latent structure in the factor matrices, the classifiers struggled to separate the classes, even after extensive tuning. However, once the decomposition provided a good separation, classification performance improved noticeably. After that, further optimising the classification models or selecting different ones had only a minor impact, even simple models like logistic regression performed just as well as more complex ones. This suggests that most of the actual "learning" happened during the tensor decomposition, not during the classification stage.
In Fig. 6.5, we even observed that *Logistic Regression* consistently yielded the best results. This futher confirms that our features (the samples factor matrix) are linearly separable, meaning a simple classifier like *Logistic Regression* is the best choice. This outcome is expected, given that the x-axis of this matrix does not represent a continuous signal, but rather discrete sample indices.

As seen in Figure 6.3, we further noticed that using a smaller rank generally led to higher average absolute correlations between the latent factors and the labels. This is likely because with fewer latent factors, the model focuses on capturing the most dominant patterns. As we increased the rank, additional, less prominent patterns were included, which lowered the average correlation.

However, despite this drop in average correlation, we saw in Table 6.5 that classification performance actually improved. This suggests that even the lower-correlation latent factors still carried useful information for the task. The key insight here is that while correlation can be a helpful indicator of decomposition quality, relying solely on the average correlation across different rank values can be misleading. Extra latent components might individually have weaker correlation, bringing the average down, but still meaningfully contribute to better classifica-

tion.

Lastly, one of our earliest insights came from preprocessing. We found that scaling the ECG signals to a consistent value range across patients was essential. Without it, the decomposition algorithm failed to identify shared patterns across samples, due to amplitude variation dominating the variance.
Other preprocessing steps, less dominant but also proved useful, were denoising and decimation. The key insight here was that these steps always improved results, hence they always had to be included in the pipeline. However, this was only the case when applied with care. When done too aggressively, the results dropped due to them removing important aspects of the signal. However, as seen in Fig 6.1, we can safely apply a decimation factor of 20 and a denoising threshold of 7.

## 7.3 Comparison with Related Work

From the related works discussed in Chapter 3, we can only make a fair comparison with studies that used the same dataset. First, we consider *Early AF Detection Using Pattern Recognition* by Wu et al. [1], outlined in Section 3.1. Like our classification task, they performed AF detection using the PTB-XL dataset [2]. However, their methodology differed significantly: they used only a single ECG lead and applied RR segmentation, resulting in simpler input data. Using Intrinsic Time-scale Decomposition and three handcrafted signal features, they achieved an accuracy of 95%, specificity of 96%, and sensitivity of 93%.

Another comparable study is *Deep Learning for ECG Classification* by Pyakillya et al. [4], discussed in Section 3.3. They also performed rhythm classification, using a deep learning approach, and achieved a best validation accuracy of approximately 86%.

Both of these results outperform our classification task, which achieved an accuracy of only 65%. However, our work demonstrates that BTTR is a promising technique, and with further development, it could become a competitive approach.

For the prediction task, comparison is more difficult, as no related work used the PTB-XL dataset. Instead, they worked with private hospital data. The first relevant study is *AF Detection on Patients During Sinus Rhythm* by Attia et al. [5], discussed in Section 3.4, which achieved an accuracy of 79%. Another is *AF Detection Using Sinus Rhythm ECGs* by Gruwez et al. [6], reviewed in Section 3.5, which achieved an accuracy of 78%. Both used similar deep learning approaches.

Again, these results are higher than ours, where we achieved 59% accuracy in the prediction task. However, we only had 42 samples in the risk class, which makes the task significantly more challenging. Despite the lower performance, we showed that AF signs are still present in SR rhythms, using an alternative and less established approach compared to neural networks, which further shows the potential of BTTR.

# Chapter 8

# Conclusion

This thesis explored how tensor decomposition can be used to extract meaningful latent features from ECG data for the classification and prediction of atrial fibrillation. While the results did not outperform state-of-the-art neural networks, the approach provided valuable insights into how tensor methods can support medical diagnosis.

## 8.1 Our experiences

At the start, we underestimated the importance of signal preprocessing. We moved too quickly into decomposition and tried to optimise it, but the results remained disappointing. Because of this, we misinterpreted several experimental outcomes, assuming certain methods didn't work, when in fact the input data was just not prepared well enough. It was only after diving deeper into preprocessing that the decomposition started to improve. From that point onward, we could finally begin to explore decomposition strategies in a meaningful way.

Another early limitation was our evaluation strategy. We initially focused only on reconstruction errors, and at one point even reconstructed the tensor using the learned factor matrices to compare the original and reconstructed signals visually. However, these signals often weren't similar, and we realised that visual similarity is not necessarily a good proxy for useful latent structure. The approach wasn't effective and risked misleading conclusions.

Once we added a classifier to the pipeline, performance was still poor, being random guessing. We tried optimising various models and parameters, but nothing worked. It was only when we started measuring correlations between the latent factors and the labels that we discovered why: there was almost no meaningful information in the decomposed features at that stage. Then we came across correlation metrics, which finally gave us a useful signal to guide optimisation. It was from this point on that we began to see actual classification performance.

This also further highlighted one of our key takeaways from the project: the decomposition step was the most decisive in the entire pipeline. Instead of spending too much time tuning classifiers, we learned that everything depends first on the quality of the features and improving decomposition had a much bigger impact than anything we did later on in the pipeline.

Looking back, the process itself was just as important as the final results, as some of these insights and approaches might help future work. This project further taught us how to approach problems when there is no clear or well-defined path. It required critical thinking, creative experimentation, and a willingness to go back and question earlier assumptions when there was no progress.

In the end, while the classification and prediction accuracy may not seem impressive on paper, the pipeline we developed showed that it is possible to extract meaningful, interpretable struc-

ture from ECG data using tensor decomposition, without deep neural networks. That alone is a powerful insight, especially for future research where interpretability, simplicity, or limited data availability are important constraints.

## 8.2 Lessons Learned

Technically, I became familiar with tensor decomposition methods and how they can be applied to waveform data. I also learned how important preprocessing is when working with such signals, and how even small signal inconsistencies can disrupt an entire pipeline. These were things I underestimated at first, but now see as essential.

In terms of research design, one of the key lessons was the importance of choosing the right evaluation metrics. For example, I initially focused on reconstruction error without considering whether that truly captured the information I needed for classification. It took several failed experiments to realise that correlation with the labels was a more relevant indicator in this context.

I also learned how critical it is to think modularly when building a pipeline. Instead of tuning everything at once, I learned to isolate each step (preprocessing, decomposition, classification) and test them independently, which saved time and clarified where problems were coming from.

Finally, I've come to see that not everything depends on how high the final accuracy is. Some of the most valuable parts of this work were the gained insights, even when things didn't work as expected.

## 8.3 Final thoughts

We started this thesis with the research question: **"How effective is Block-Term Tensor Regression in detecting and predicting atrial fibrillation from multi-lead ECG, and how can its pipeline be optimised for best performance?"**

Looking back, we can conclude that for now, Block-Term Tensor Regression is not as effective as deep neural networks for this task. However, the method shows potential. We have shown that it is possible to apply this approach to both the detection and prediction of atrial fibrillation from ECG data, and that it can produce useful results.

One of the main advantages we experienced is that the model offers some level of interpretability through the decomposition visualisations. While we cannot yet draw concrete conclusions from these visualisations, they give an idea of what might be possible in the future. Another benefit is that BTTR seemed to perform reasonably well even with a limited amount of data. For example, in the prediction task we only had 42 positive samples, but the model still reached an accuracy of 59%.

In summary, while BTTR does not currently outperform deep learning models in terms of raw accuracy, it offers useful properties such as interpretability and better performance with small datasets, while still giving results. This makes it an interesting approach for future research.

## 8.4 Future work

Seeing the results in accuracy, recall and precision, especially for our prediction task, it is clear that there remains significant room for improvement.
First of all, we recommend applying this methodology to a complete Block-Term Decomposition in the future, once a well-defined implementation becomes available. The same approach can be reused, as both decomposition techniques are similar in intuition and operation, with BTD

offering a higher-dimensional representation by capturing more relationships between the tensor modes.

Furthermore, with regard to the prediction task, future work could explore applying this method to a clinical dataset within a hospital setting, where a larger number of patients with AF also have SR rhythms recorded. This would allow for a richer dataset and potentially lead to more reliable results.

Another possible direction for future work is the use of RR segmentation, which was applied in several related works. This would allow the model to work with individual heartbeats instead of the full 10-second ECG segment. One way to do this is to use just one beat per sample, which simplifies the data and could help focus on beat-specific patterns. Alternatively, RR segmentation could still be used while keeping multiple beats from the same sample together. These beats could then be treated as an extra dimension in the input tensor. This might help structure the data more clearly, as it allows the model to look at separate heartbeats within the context of the full rhythm, without needing to completely reconstruct the entire 10 seconds at once.

# Bibliography


[1] X. Wu, Y. Zheng, Y. Che, and C. Cheng, "Pattern recognition and automatic identification of early-stage atrial fibrillation," *Expert Systems with Applications*, vol. 158, p. 113560, 2020.

[2] P. Wagner, N. Strodthoff, R.-D. Bousseljot, D. Kreiseler, F. I. Lunze, W. Samek, and T. Schaeffter, "Ptb-xl: A large publicly available ecg dataset," *Scientific Data*, vol. 7, no. 1, pp. 1–15, 2020.

[3] M. Kachuee, S. Fazeli, and M. Sarrafzadeh, "Ecg heartbeat classification: A deep transferable representation," in *2018 IEEE international conference on healthcare informatics (ICHI)*. IEEE, 2018, pp. 443–444.

[4] B. Pyakillya, N. Kazachenko, and N. Mikhailovsky, "Deep learning for ecg classification," in *Journal of physics: conference series*, vol. 913, no. 1. IOP Publishing, 2017, p. 012004.

[5] Z. I. Attia, P. A. Noseworthy, F. Lopez-Jimenez, S. J. Asirvatham, A. J. Deshmukh, B. J. Gersh, R. E. Carter, X. Yao, A. A. Rabinstein, B. J. Erickson *et al.*, "An artificial intelligence-enabled ecg algorithm for the identification of patients with atrial fibrillation during sinus rhythm: a retrospective analysis of outcome prediction," *The Lancet*, vol. 394, no. 10201, pp. 861–867, 2019.

[6] H. Gruwez, M. Barthels, P. Haemers, F. H. Verbrugge, S. Dhont, E. Meekers, F. Wouters, D. Nuyens, L. Pison, P. Vandervoort *et al.*, "Detecting paroxysmal atrial fibrillation from an electrocardiogram in sinus rhythm: external validation of the ai approach," *Clinical Electrophysiology*, vol. 9, no. 8_Part_3, pp. 1771–1782, 2023.

[7] A. Goldberger, L. Amaral, L. Glass, J. Hausdorff, P. C. Ivanov, R. Mark, and H. E. Stanley, "Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000.

[8] P. Wagner, N. Strodthoff, R. Bousseljot, W. Samek, and T. Schaeffter, "PTB-XL, a large publicly available electrocardiography dataset (version 1.0.3)," https://doi.org/10.13026/kfzx-aw45, 2022, physioNet.

[9] World Health Organization. (2021, jun) Cardiovascular diseases (cvds). World Health Organization. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds)

[10] Mayo Clinic. (2024, may) Atrial fibrillation - symptoms and causes. Mayo Foundation for Medical Education and Research. [Online]. Available: https://www.mayoclinic.org/diseases-conditions/atrial-fibrillation/symptoms-causes/syc-20350624

[11] M. Lemoine, L. Rottner, and P. Kirchhof, "Asymptomatic atrial fibrillation: the tasks ahead," *European Heart Journal*, vol. 46, no. 13, pp. 1203–1205, 02 2025. [Online]. Available: https://doi.org/10.1093/eurheartj/ehae835

[12] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[13] A. A. Rontogiannis, E. Kofidis, and P. V. Giampouras, "Block-term tensor decomposition: Model selection and computation," *IEEE Journal of Selected Topics in Signal Processing*, vol. 15, no. 3, pp. 464–475, 2021.

[14] R. Vandebril, L. Sorber, and L. D. Lathauwer, "Block term decomposition," https://www.tensorlab.net/doc/btd.html, n.d.

[15] D. R. Cox, "The regression analysis of binary sequences," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 20, no. 2, pp. 215–232, 1958.

[16] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1. IEEE, 1995, pp. 278–282.

[17] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 785–794. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939785

[18] Y. Sattar and L. Chhabra, *Electrocardiogram.* Treasure Island (FL): StatPearls Publishing, 2023, updated 2023 Jun 5. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK549803/

[19] Z. Nesheiwat, A. Goyal, and M. Jagtap, *Atrial Fibrillation.* Treasure Island (FL): StatPearls Publishing, 2023, updated 2023 Apr 26. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK526072/

[20] A. Işın and S. Özdalili, "Cardiac arrhythmia detection using deep learning," *Procedia Computer Science*, vol. 120, pp. 268–275, 01 2017.

[21] A. Faes, F. Camarrone, and M. M. Van Hulle, "Single finger trajectory prediction from intracranial brain activity using block-term tensor regression with fast and automatic component extraction," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[22] D. Hong, T. G. Kolda, and J. A. Duersch, "Generalized canonical polyadic tensor decomposition," *SIAM Review*, vol. 62, no. 1, pp. 133–163, 2020.

[23] L. Souquières, C. Prissette, S. Maire, and N. Thirion-Moreau, "A parallel strategy for an evolutionary stochastic algorithm: application to the cp decomposition of nonnegative n -th order tensors," in *2020 28th European Signal Processing Conference (EUSIPCO).* Amsterdam, Netherlands: IEEE, 2020, pp. 1350–1354.

[24] J. Zhao, S. Zheng, H. Huo, M. Gong, T. Zhang, and L. Qu, "Fast weighted cp decomposition for context-aware recommendation with explicit and implicit feedback," *Expert Systems with Applications*, vol. 204, p. 117591, 2022.

[25] L. Zou, X. Chen, and Z. J. Wang, "Underdetermined joint blind source separation for two datasets based on tensor decomposition," *IEEE Signal Processing Letters*, vol. 23, no. 5, pp. 673–677, 2016.

[26] European Committee for Standardization, "Standard communications protocol for computer-assisted electrocardiography (scp-ecg)," CEN ENV 1064:1995, Brussels, 1995, european prestandard.

[27] S. Luo and P. Johnston, "A review of electrocardiogram filtering," *Journal of electrocardiology*, vol. 43, no. 6, pp. 486–496, 2010.

[28] A. Gacek, "Preprocessing and analysis of ecg signals – a self-organizing maps approach," *Expert Systems with Applications*, vol. 38, no. 7, pp. 9008–9013, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417411001394

[29] M. Bashiri, "utils.py — tensor decomposition in python," https://github.com/mohammadbashiri/tensor-decomposition-in-python/blob/master/utils.py, 2019.

[30] J. Queiroz, L. Azoubel, and A. Barros, "Support system for classification of beat-to-beat arrhythmia based on variability and morphology of electrocardiogram," *EURASIP Journal on Advances in Signal Processing*, vol. 2019, no. 1, p. 16, 2019. [Online]. Available: https://doi.org/10.1186/s13634-019-0613-9

[31] S. Hargittai, "Efficient and fast ecg baseline wander reduction without distortion of important clinical information," in *2008 Computers in Cardiology*. IEEE, 2008, pp. 841–844.

[32] A. Singhal, P. Singh, B. Fatimah, and R. B. Pachori, "An efficient removal of power-line interference and baseline wander from ecg signals by employing fourier decomposition technique," *Biomedical Signal Processing and Control*, vol. 57, p. 101741, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1746809419303222

[33] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[34] B. Salsekar and A. Wadhwani, "Filtering of ecg signal using butterworth filter and its feature extraction," *Int. J. Eng. Sci. Technol*, vol. 4, no. 2, 2012.

[35] Y.-W. Bai, W.-Y. Chu, C.-Y. Chen, Y.-T. Lee, Y.-C. Tsai, and C.-H. Tsai, "Adjustable 60hz noise reduction by a notch filter for ecg signals," in *Proceedings of the 21st IEEE Instrumentation and Measurement Technology Conference (IEEE Cat. No. 04CH37510)*, vol. 3. IEEE, 2004, pp. 1706–1711.

[36] M. Alfaouri and K. Daqrouq, "Ecg signal denoising by wavelet transform thresholding," *American Journal of applied sciences*, vol. 5, no. 3, pp. 276–281, 2008.

[37] Y. Eminaga, A. Coskun, S. A. Moschos, and I. Kale, "Low complexity all-pass based polyphase decimation filters for ecg monitoring," in *2015 11th Conference on Ph. D. Research in Microelectronics and Electronics (PRIME)*. IEEE, 2015, pp. 322–325.

[38] C. Shannon, "Communication in the presence of noise," *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, jan 1949. [Online]. Available: https://doi.org/10.1109/jrproc.1949.232969

[39] P. M. R. de Oliveira, J. d. M. Goulart, C. A. R. Fernandes, and V. Zarzoso, "Blind source separation in persistent atrial fibrillation electrocardiograms using block-term tensor decomposition with löwner constraints," *IEEE Journal of Biomedical and Health Informatics*, vol. 26, no. 4, pp. 1538–1548, 2021.

[40] J. Kossaifi, Y. Panagakis, A. Anandkumar, and M. Pantic, "Tensorly: Tensor learning in python," *Journal of Machine Learning Research*, vol. 20, no. 26, pp. 1–6, 2019. [Online]. Available: http://jmlr.org/papers/v20/18-277.html

[41] S. Zhou, N. X. Vinh, J. Bailey, Y. Jia, and I. Davidson, "Accelerating online cp decompositions for higher order tensors," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1375–1384.

[42] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[43] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[44] J. H. Zar, "Spearman rank correlation," *Encyclopedia of Biostatistics*, vol. 7, 2005.

[45] D. Freedman, R. Pisani, and R. Purves, "Statistics (international student edition)," *Pisani, R. Purves, 4th edn. WW Norton & Company, New York*, 2007.

[46] S. I. Sabilla, R. Sarno, and K. Triyana, "Optimizing threshold using pearson correlation for selecting features of electronic nose signals," *Int. J. Intell. Eng. Syst*, vol. 12, no. 6, pp. 81–90, 2019.

[47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[48] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip *et al.*, "Top 10 algorithms in data mining," *Knowledge and information systems*, vol. 14, no. 1, pp. 1–37, 2008.

[49] G. R. Lee, R. Gommers, F. Waselewski, K. Wohlfahrt, and A. O. Leary, "Pywavelets: A python package for wavelet analysis," *Journal of Open Source Software*, vol. 4, no. 36, p. 1237, 2019. [Online]. Available: https://doi.org/10.21105/joss.01237

[50] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, p. 357–362, 2020.

[51] W. McKinney *et al.*, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, vol. 445. Austin, TX, 2010, pp. 51–56.

[52] M. Waskom, O. Botvinnik, D. O'Kane, P. Hobson, S. Lukauskas, D. C. Gemperline, T. Augspurger, Y. Halchenko, J. B. Cole, J. Warmenhoven, J. de Ruiter, C. Pye, S. Hoyer, J. Vanderplas, S. Villalba, G. Kunter, E. Quintero, P. Bachant, M. Martin, K. Meyer, A. Miles, Y. Ram, T. Yarkoni, M. L. Williams, C. Evans, C. Fitzgerald, Brian, C. Fonnesbeck, A. Lee, and A. Qalieh, "mwaskom/seaborn: v0.8.1 (september 2017)," Sep. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.883859

[53] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in science & engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[54] Z. Zhao and A. E. W. Johnson, "WFDB Python Toolbox," https://github.com/MIT-LCP/wfdb-python, 2018.