Signal Processing Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
https://sps.ewi.tudelft.nl/

SPS-2025

M.Sc. Thesis

End-to-End Embedded Machine Learning for In-Ear PPG Peak Detection

Sebastian Speekenbrink

Abstract

Medical monitoring technologies have gained increasing importance in recent years. Among emerging wearables, in-ear sensing offers a promising alternative to wrist-based devices due to its stable environment and proximity to major arteries, with machine-learning (ML) models showing potential to improve signal analysis performance in this domain, although their design and implementation often lack systematic methodology and reproducibility. This thesis aims to address these gaps by designing an end-to-end in-ear cardiac monitoring system, from custom hardware and dataset collection to the development of a reproducible machine-learning framework for peak detection suitable for embedded deployment. A custom-fit, multi-location in-ear photoplethysmography (PPG) sensing system was developed to collect a multi-activity dataset with a ground-truth electrocardiogram (ECG) reference, enabling systematic evaluation of different Convolutional Neural Network (CNN) architectures for embedded purposes. Results show that signal quality, and thus model performance, strongly depends on sensor placement, with the deep external auditory meatus providing the best signals, followed by the concha. The systematic architecture exploration further revealed consistent design patterns associated with higher accuracy, enabling efficient peak detection with strong ECG correlation. Overall, this work establishes a standardised framework for automatically identifying optimal embedded model architectures for in-ear PPG analysis. Key limitations include the single-subject dataset, computational constraints during model training, and limited final on-device validation.

Faculty of Electrical Engineering, Mathematics and Computer Science

Delft University of Technology

End-to-End Embedded Machine Learning for In-Ear PPG Peak Detection

Thesis

submitted in partial fulfillment of the
requirements for the degree of

Master of Science

in

Embedded Systems

by

Sebastian Speekenbrink
born in Delft, Netherlands

This work was performed in:

Signal Processing Systems Group
Department of Microelectronics
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Delft University of Technology
Department of
Microelectronics

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"End-to-End Embedded Machine Learning for In-Ear PPG Peak Detection"** by **Sebastian Speekenbrink** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 18 Dec, 2025

Chairman:

dr.ir. R.C. Hendriks

Advisor:

dr. ir. J.C. Haartsen

Committee Members:

dr. D.M.J. Tax

dr. C. Gao

ir. A. Boru

Abstract

Medical monitoring technologies have gained increasing importance in recent years. Among emerging wearables, in-ear sensing offers a promising alternative to wrist-based devices due to its stable environment and proximity to major arteries, with machine-learning (ML) models showing potential to improve signal analysis performance in this domain, although their design and implementation often lack systematic methodology and reproducibility. This thesis aims to address these gaps by designing an end-to-end in-ear cardiac monitoring system, from custom hardware and dataset collection to the development of a reproducible machine-learning framework for peak detection suitable for embedded deployment. A custom-fit, multi-location in-ear photoplethysmography (PPG) sensing system was developed to collect a multi-activity dataset with a ground-truth electrocardiogram (ECG) reference, enabling systematic evaluation of different Convolutional Neural Network (CNN) architectures for embedded purposes. Results show that signal quality, and thus model performance, strongly depends on sensor placement, with the deep external auditory meatus providing the best signals, followed by the concha. The systematic architecture exploration further revealed consistent design patterns associated with higher accuracy, enabling efficient peak detection with strong ECG correlation. Overall, this work establishes a standardised framework for automatically identifying optimal embedded model architectures for in-ear PPG analysis. Key limitations include the single-subject dataset, computational constraints during model training, and limited final on-device validation.

Acknowledgments

I would like to start by thanking my professor dr.ir. R.C. Hendriks for providing the possibility to start a MSc. thesis under his guidance back in 2023 before my exchange in Australia. After coming back from this exchange, he recommended me to get in contact with dr. ir. J.C. Haartsen, something that proved to be very valuable and arguably the most impactful decision in this whole thesis project.

Furthermore, I would like to acknowledge all the effort performed by dr. ir. J.C. Haartsen, F. Hooijschuur and the other members of Dopple, the company that provided support in designing and manufacturing of the in-ear sensing setup. Without you, this thesis would not have been able to reach its current form.

Moreover, I would like to show gratitude to my colleagues at UbiOps for giving me the freedom and flexibility to perform this research in parallel with my professional work, as this support allowed me to finish this thesis successfully.

At last, I would like to extend my appreciation to my friends and family, the people who stood by my side and showed support that kept me going.

Sebastian Speekenbrink
Delft, The Netherlands
18 Dec, 2025

Contents

List of Figures

List of Tables

Introduction 1

The significance and use of medical monitoring have surged in recent years, enabling the early detection of health conditions by tracking various health indicators. Consequently, developing systems that facilitate such monitoring is essential, allowing for timely interventions with potentially improved patient outcomes and reducing the burden on the healthcare system [1], [2].

Various methods have been developed to monitor a broad spectrum of health indicators, among which vital signs. These vital signs, including body temperature, pulse rate, respiratory rate and blood pressure, provide an objective measurement of the essential physiological functions of the human body. By emphasizing these vital signs, medical monitoring could be immensely valuable to monitor the well-being of an individual [3].

One prevalent method of medical monitoring is wearables; sensors that one can wear on their body that will monitor one or more health indicators. These wearables differ on many different aspects, e.g. body location, measurements performed and materials used [4], [5].

An interesting body location that has not been widely adopted yet for wearables are inside the ear. In-ear wearables offer a unique opportunity due to their close proximity to an artery and relatively stable environment. Wrist-worn wearables are much more susceptible to motion artefacts and external temperature changes, which makes in-ear measurements a rather interesting field of study. Despite these important advantages, research and real-life appliances of in-ear wearables are still in their early stages [6].

Among the physiological parameters accessible through in-ear sensing, cardiac activity holds particular relevance. Cardiac activity can be evaluated through several measurable metrics, including heart rate (HR) and heart rate variability (HRV), which are key indicators of cardiovascular health and autonomic nervous system balance. Accurately identifying these metrics under varying conditions requires robust signal processing and reliable extraction methods, forming the foundation for further developments in in-ear monitoring [7], [8].

Different sensing methods are available for extracting cardiac activity. The gold standard for cardiac monitoring is the electrocardiogram (ECG), which records the electrical potentials generated by the heart. In wearable systems, however, photoplethysmography (PPG) is an accessible alternative due to its compact optical setup and straightforward integration in miniature devices. PPG estimates cardiac activity by measuring changes in light absorption related to blood volume variations, enabling reliable heart rate estimation with minimal hardware complexity. Other approaches, such as ballistocardiography, thermometry, or acoustic and piezoelectric sensing, have also been explored for heart rate estimation, though these approaches are generally less mature or more sensitive to motion artefacts [9], [10].

However, despite promising results, the accuracy and robustness of in-ear cardiac sensing are often hindered by motion artefacts, sensor placement variability, and physiological differences among individuals. These limitations underline the need for advanced signal-processing or learning-based methods capable of generalizing across conditions [11], [12].

Artificial intelligence (AI) represents another important trend in healthcare technology. Machine learning (ML) techniques are increasingly applied to tasks such as medical diagnostics, predictive modeling, and personalised health monitoring. Within the context of wearable devices, AI could play a crucial role in handling noisy signals, extracting meaningful features, and enabling robust prediction of health outcomes [13], [14].

Although these techniques offer significant potential, the process of designing effective machine learning models for biomedical signals remains highly empirical. Selecting appropriate architectures, kernel sizes, or dilation factors often depends on prior experience or domain-specific heuristics rather than clear design principles. This lack of systematic understanding can make it difficult to determine which configurations are most suitable for specific hardware or signal conditions, particularly when operating under resource constraints [15].

Several studies have attempted to address these limitations by exploring resource-aware or automated model optimisation for physiological signal analysis. However, many of these efforts still lack methodological transparency, with datasets, model configurations, or training procedures not publicly released. Moreover, the depth of optimisation is typically restricted with grid search often applied only to a limited set of hyperparameters, while the overall model architecture remains fixed and cannot be systematically explored. As a result, it is difficult to reproduce findings or to isolate which design decisions drive performance [16], [17].

Despite the potential of these approaches, the understanding and application of machine learning remain challenging for many researchers and developers outside the AI domain. Building, training, and validating ML models requires expertise in data preprocessing, model design, and hyperparameter optimisation. These skills are not always present among domain specialists in fields such as biomedical engineering or healthcare technology. This knowledge gap often limits the adoption of ML in applied research, even when the potential benefits are clear.

An additional consideration in the development of AI (for healthcare), is the extent to which research, and the work behind this research, has been made openly available. Historically speaking, AI research has not been providing enough documentation for reproducibility. Open-source attempts, such as sharing datasets, coding practices and engineering methods, enable other researchers to build upon existing research, instead of spending time and effort in recreating systems. This in turn could accelerate scientific progress with a range of benefits [18], [19].

Beyond its technical motivation, this research also carries broader societal relevance. By enabling continuous, non-invasive monitoring of cardiac activity in a compact and personally adaptable form factor, in-ear sensing can advance preventive and personalised healthcare. Furthermore, developing reproducible machine-learning frameworks for embedded health analytics supports transparent and accessible digital health solutions.

Taking this all in, a clear gap emerges at the intersection of in-ear sensing and machine learning. While in-ear PPG offers strong physiological advantages, its practical use is limited by motion artefacts and variations in placement. Addressing these issues requires robust ML methods, yet remain difficult to adopt in practice due to the lack of a transparent, reproducible design framework taking in different resource constraints. In particular, no accessible methodology is available that enables researchers to systematically optimise models for the constraints of in-ear hardware.

To address the gaps in the currently available research, the main aim of this thesis is to investigate the feasibility of reliable cardiac activity estimation from PPG signals acquired inside the ear canal and to develop a transparent, reproducible framework for analysing and deploying ML models on embedded systems. To achieve this, the following list of objectives was determined:

- Design and develop a custom-fit in-ear measurement system:
  - Create a replaceable ear-shell design that can be detached from a separate base unit containing the processing and communication electronics, with the replaceable ear-shell integrating a fixed-placement acceleration sensor and a repositionable PPG sensor at multiple ear locations (tragus, concha, near- and inside the external auditory meatus).
  - Ensure wireless data acquisition and multi-wavelength sensing.
- Collect and organise a physiological dataset:
  - Record synchronised PPG, accelerometer, and ECG reference data under varying activity conditions (seated, walking).
  - Implement automated and manual alignment methods to accurately synchronise sensor data and ECG ground truth.
  - Evaluate signal quality across different in-ear placements using qualitative and quantitative metrics.
- Develop a configurable and reproducible machine-learning framework:

  - Build an automated pipeline for data preprocessing, annotation, and segmentation of PPG signals.
  - Implement multiple configurable 1D CNN architectures that support systematic hyperparameter and architectural exploration.
  - Facilitate re-use for future research/reproducibility.

- Evaluate and analyse model performances:
  - Quantify performance of different models.
  - Investigate the influence of different model parameters on peak detection accuracy.
  - Examine PPG sensor location impact on model performance.

- Deploy and validate the optimised model on an embedded microcontroller:
  - Convert the selected model to an embedded-compatible format.
  - Validate functional equivalence and benchmark performance.

In line with these objectives, this thesis delivers an end-to-end in-ear monitoring system, comprising custom sensor hardware, a reproducible machine-learning framework for cardiac peak detection with systematic model analysis, and embedded validation of the trained model for deployment on resource-constrained devices. Rather than prescribing a single model, this thesis introduces a configurable framework that automatically selects the most suitable architecture for the available dataset and specified hardware constraints. As PPG characteristics and embedded constraints vary across applications, this adaptive approach ensures that each resulting model is optimised for the specific sensing conditions instead of relying on a one-size-fits-all design. A preliminary overview of the research methodology can be seen in Figure 1.1.

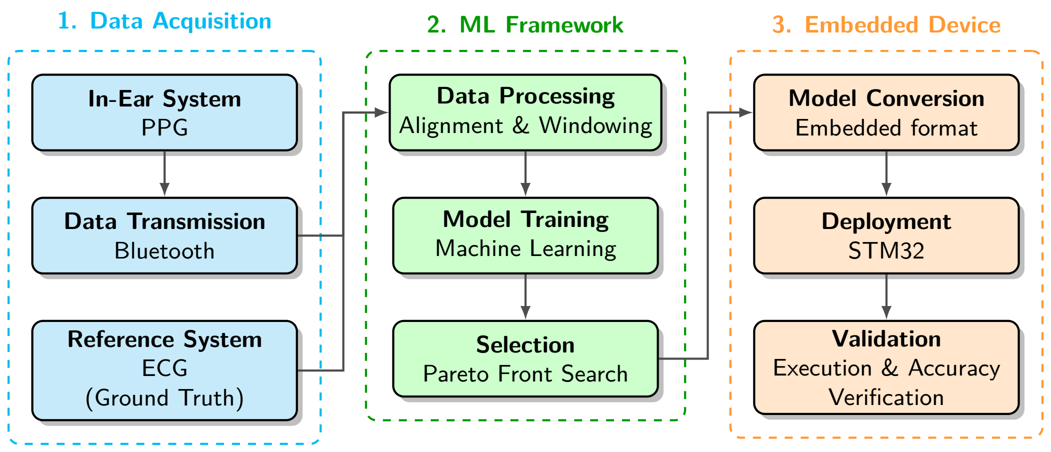

1. Data Acquisition
In-Ear System PPG
Data Transmission Bluetooth
Reference System ECG (Ground Truth)
2. ML Framework
Data Processing Alignment & Windowing
Model Training Machine Learning
Selection Pareto Front Search
3. Embedded Device
Model Conversion Embedded format
Deployment STM32
Validation Execution & Accuracy Verification

Figure 1.1: Overview of the methodology used in this thesis

The remainder of this thesis is structured as follows. chapter 2 outlines the methodology used to realise the objectives. Chapter 3 presents the corresponding obtained results. Chapter 4 discusses the results, interprets their implications, considers the limitations of this research and explores directions for future research. At last, chapter 5 summarises the main contributions and concludes the thesis.

All relevant code will be published to the corresponding GitHub repository.

Methodology 2

This chapter describes the methodology used to design and develop the hardware and its corresponding software, the dataset collected using this hardware, the generation of different machine learning models, and their deployment on an embedded device for validation. The chapter is therefore divided into multiple subsections, each addressing a specific part of the process, and is concluded with an overview of the different experiments performed.

2.1 Hardware

To quantify cardiac activity, an experimental hardware setup was developed together with Dopple [20]. The core of this setup is a custom-fit in-ear module incorporating a photoplethysmography (PPG) sensor. This sensor measures blood volume fluctuations in the ear's vasculature, from which cardiac activity can be derived. PPG was selected as the sensing modality because it can be easily integrated into an in-ear system, requiring only a single measurement site and offering a much smaller form factor compared to ECG sensors. The hardware system is composed of two primary elements: the custom-fit ear shell and the integrated electronics.

2.1.1 Mechanical Design and Fabrication

The custom-fit ear shell was fabricated using two distinct methodologies to ensure an optimal and secure fit for the user. As both approaches resulted in equivalent fit characteristics, the fabrication method was selected based on availability and convenience, with no comparative analysis performed.

The first method involved the direct digital acquisition of the ear's geometry using an Otoscan handheld 3D ear scanner developed by Natus Medical [21]. This process generated a precise 3D digital model of the ear canal and concha, which could directly be used to create a new 3D model in which the PPG sensor could be integrated into the surface of the ear shell for optimal sensor-to-skin contact. This new 3D model could then be used in a future 3D printing stage.

The second method utilised a traditional physical impression technique. An ear impression was created using Otoform, a two-component silicone casting compound developed by Dreve Otoplastik [22]. This physical mould, which accurately captures the topology of the ear, was subsequently digitised using a 3D scanner. The final custom-fit shell was then manufactured based on this digital model.

2.1.2 Electronic Components and System Integration

To ensure reliable signal acquisition and experimental flexibility, several key requirements were defined for the sensing system. These requirements guided both the hardware design and component selection for the in-ear measurement setup:

- **Interchangeable earbud design:** The earbud must be swappable to facilitate measurements at multiple in-ear locations, allowing for a comparison of different sensor placements across ear positions.
- **Multi-wavelength PPG:** The system should support multiple optical wavelengths to allow the usage of different wavelengths (if needed) for different ear sensor locations. The green colour with a wavelength of around 525 nm must be included, as it has been shown that this wavelength is affected by motion artifacts to a lesser degree than other commonly used wavelengths for PPG and is applicable for different skin types [23], [24].
- **PPG sampling rate:** A minimum sampling rate of 100 Hz is required when no interpolation is used to ensure accurate pulse timing, while 50 Hz is only feasible when applying interpolation to refine pulse locations, as shown in [25].
- **Compact form factor:** A compact form factor is required for the PPG and acceleration sensor to fit within the custom-made ear mould, with the exact allowable dimensions depending on the resulting ear mould.
- **Acceleration sensing:** An integrated accelerometer is required for motion tracking, signal alignment, and potential artefact analysis, operating at a sampling frequency at least equal to that of the PPG sensor.
- **Wireless data transmission:** The system must support wireless communication to allow freedom of movement during recordings while maintaining data integrity.
- **Recording duration:** The system should support continuous data acquisition for a minimum of two hours to capture extended physiological sessions without interruption.

These requirements resulted in a modular electronic system, consisting of a main unit and a swappable ear sensor piece. The main unit houses the central processing and communication components, while the ear sensor piece contains the sensing elements, enabling the testing of different sensor placements within the same system architecture.

The main unit is powered by a rechargeable battery and is controlled by a Renesas DA14695 System-on-Chip (SoC) [26]. This microcontroller was selected due to its integrated Bluetooth functionality, adequate processing capabilities and its availability from a previous project iteration, which streamlined the development process. The main unit can be seen in Figure 2.1. The main unit is integrated into a headband for easy wearability, with the connector for attaching the interchangeable earbud clearly visible.

The interchangeable earbud integrates two key sensors:

- **PPG Sensor:** A Maxim Integrated MAXM86161 optical sensor is used for PPG. This sensor was chosen for its optimisation for in-ear applications [27]. Its compact form factor (2.9 mm x 4.3 mm x 1.4 mm), three programmable

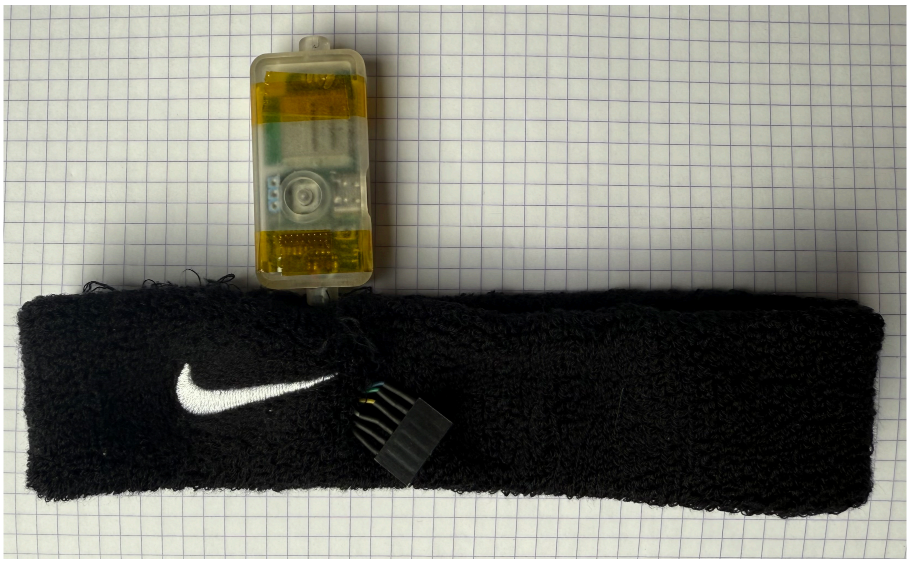


Figure 2.1: Main unit with connector for interchangeable earbud integrated into wearable headband (developed by Dopple).

green, IR and red LED drivers with centroid wavelengths of 520-535, 880 and 660 nm, respectively and a sampling frequency range between 8 and 4096 sps satisfied all requirements. Furthermore, its low power consumption and error rejection characteristics made it an ideal sensor for integration into the in-ear device.

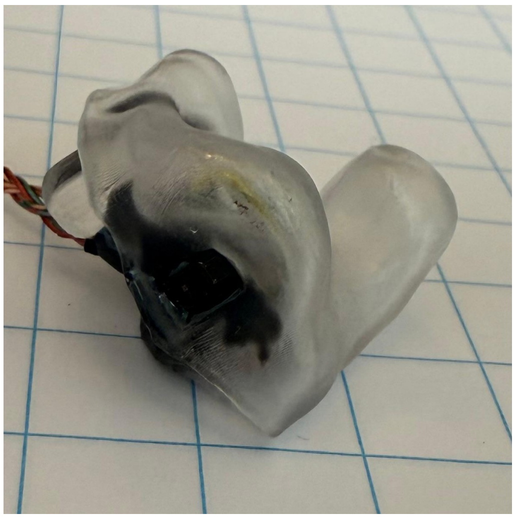
- **Accelerometer:** A BMA580 3-axis accelerometer [28] is included for motion tracking. This acceleration sensor has the ability to use sample rates from 1.5625 Hz to 6.4 kHz and an extremely compact form factor (1.2 mm x 0.8 mm x 0.55 mm), which makes it a perfect fit for the set requirements.

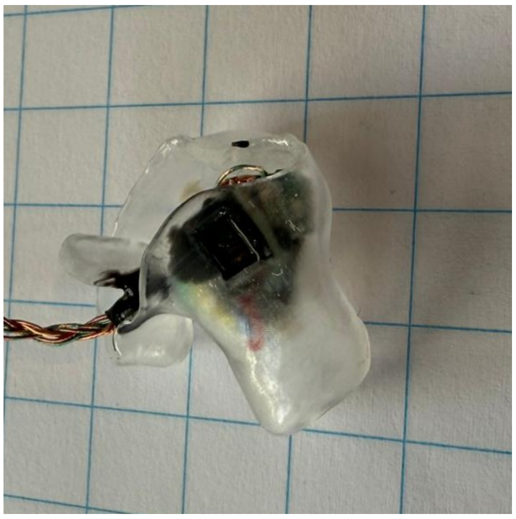
2.1.3 Sensor Placements

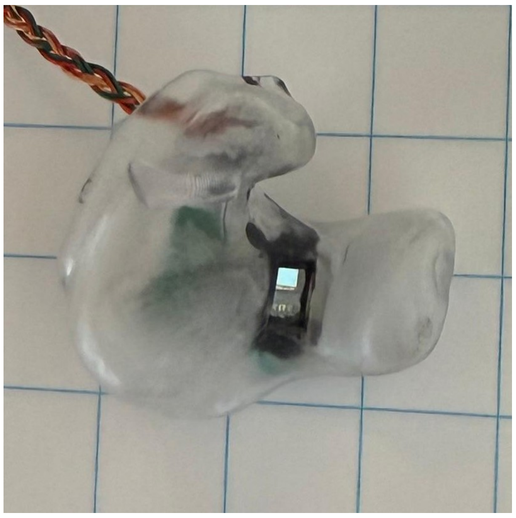
The PPG sensor in the earbud is placed in one of four different locations, depending on the specific module being tested. These four locations are against the tragus (Figure 2.2b), deep inside the external auditory meatus facing the eyes (Figure 2.2d), the concha (Figure 2.2a) and shallow inside the external auditory meatus (Figure 2.2c). The performance associated with each placement will be analysed in detail in later sections, followed by recommendations regarding the most suitable location for different goals.

2.2 Data Acquisition and Processing

The system employs an event-driven architecture to minimise power consumption. Rather than continuously polling the sensors, the MCU remains in a low-power state until it is awakened by a hardware interrupt. This process is initiated by the PPG sensor's internal First-In, First-Out (FIFO) buffer, which sends an interrupt after a sample has been collected. After an interrupt has been received from the PPG


(a) Sensor placed in the concha


(b) Sensor placed at the tragus


(c) Sensor placed shallowly in the external auditory meatus

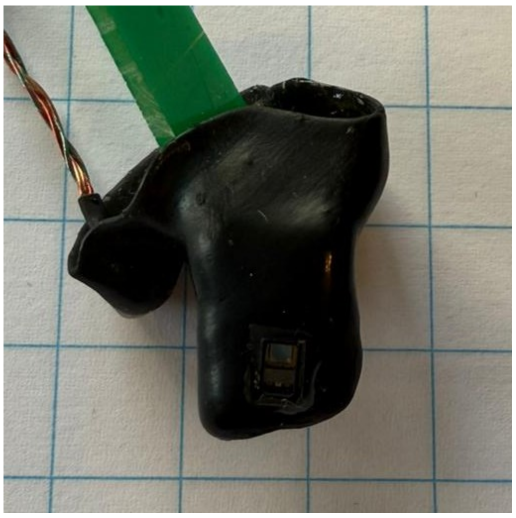

(d) Sensor placed deeply in the external auditory meatus

Figure 2.2: Overview of the four in-ear PPG sensor placements.

sensor, the system retrieves the measured values from the PPG and acceleration sensor. These values are then sent over a Bluetooth connection.

The corresponding firmware implementation was developed in collaboration with Dopple [20]. The codebase was largely based on existing internal examples from their library, which were adapted for the current setup. As a result, certain implementation shortcuts were retained to enable fast integration and prototyping.

2.2.1 Sensor Configuration and Synchronisation

The sensors are configured with specific sampling rates to capture relevant physiological and motion data effectively.

- The PPG sensor (MAXM86161) is configured to sample its four channels

(Green, IR, Red, and Ambient) at a rate of 100 Hz. Each reading is a 19-bit value (stored in a 32-bit unsigned integer), with the maximum value corresponding to a software-set maximum photodiode input current. The maximum is set in this research to $16\mu A$.

- The accelerometer (BMA580) operates at 400 Hz with a range of $\pm 2g$. Each axis is represented as a signed 16-bit value in the range $[-32{,}768, 32{,}767]$.

The decision on the specific sampling rate of the acceleration sensor relative to the PPG sensor can be guided by the maximum allowable temporal offset between samples of both signals. This offset is fully determined by the acceleration frequency, with the formula shown in Equation 2.1. In this scenario, as the PPG signal is sampled at 100 Hz, each PPG sample must have a temporally corresponding acceleration value. An acceleration sampling rate lower than the PPG sampling rate could introduce uncertainty during signal alignment, where multiple PPG samples might correspond to a single acceleration measurement. Therefore, a minimum acceleration sampling frequency of 100 Hz is required to ensure temporal correspondence between PPG and acceleration data, which is essential for stages discussed later in this work.

$$\Delta t_{\text{max}} = \frac{1}{f_{acc}} \tag{2.1}$$

2.2.2 Data Transmission Protocol

To facilitate the ability to measure in different environments, the data will be transmitted wirelessly.

2.2.2.1 Packet Structure.

The data is formatted into a custom packet structured as an ASCII string of comma-separated values (CSV), terminated by a newline character (`\n`). This human-readable format allows easy and direct inspection of the individual values when received, without requiring custom software to unpack the data. Each packet contains seven fields, representing a single measurement:

```
[Green PPG],[IR PPG],[Red PPG],[Ambient Light],[Accel X],[Accel Y],[Accel Z]\n
```

An example packet could thus be:

```
4140,5371,9886,175,2374,-11626,-11619\n
```

While another viable approach would be to transmit the raw byte structure directly over the Bluetooth connection, this option was not implemented, as it would complicate direct readability and verification during data acquisition. The chosen ASCII-based representation thus prioritises transparency and ease of debugging during development and testing, at the cost of a slightly higher transmission overhead, which is more than acceptable given the current size of the data stream relative to the available bandwidth.

2.2.2.2 Batched Transmission.

To improve system efficiency and power usage, individual data packets are transmitted in batches of 4 (or when the transmission buffer is full) instead of individually. For data reception and logging, the Renesas SmartConsole mobile application was used on a smartphone to receive the data from the sensing system.

2.3 Dataset

Using the developed experimental in-ear measurement system, a dedicated dataset was recorded to evaluate the performance of the PPG signals obtained from different ear sensor locations against a reliable cardiac ground truth. The dataset was specifically designed to enable both quantitative and qualitative analyses of signal quality, temporal alignment, and model development between different sensor locations. Each recording session comprised synchronised PPG, acceleration, and ECG data streams, acquired under controlled and repeatable conditions. Different activity scenarios were included to capture the influence of motion on signal characteristics, while repeated recordings across different placements provided comparative measurements. The following subsections describe the dataset structure, signal quality evaluation, ground-truth acquisition, alignment procedures, validation process and final data storage format.

2.3.1 Dataset overview

For each sensor location, two distinct activity conditions were measured, each with a minimum total recording duration accumulated over a single or multiple sessions: a seated condition of approximately 20 minutes, and a motion condition (walking back and forth in a straight line) of approximately 10 minutes. Recordings were obtained at four different sensor locations (outlined in 2.1.3) within the ear to enable a comparative analysis of location-dependent PPG performance. For the deep external auditory ear meatus sensor location, additional recordings were collected to enable a more comprehensive analysis and improve robustness when evaluating those specific placements. These recordings were not used in the comparative analysis between different ear locations, but only for a single-sensor model performance analysis. An overview of the dataset composition is given in Table 2.1.

Table 2.1: Dataset composition across different recording locations with PPG sensor at $f_s \approx 100$.

Location	**Seated samples**	**Moving samples**
Tragus	[135564]	[66184]
External auditory meatus (Deep)	[321598]	[66264]
External auditory meatus (Shallow)	[135552]	[67582]
Concha	[121940]	[63732]

2.3.2 PPG Signal quality measurement

In order to quantify the signal quality of a measured PPG signal, the Skewness factor is used. [29] identified this metric as a reliable indicator of PPG waveform quality, where a positive skewness value (> 0) typically corresponds to a proper PPG signal. The formula for the skewness factor is outlined in Equation 2.2(from [30]).

$$g_1 = \frac{m_3}{m_2^{3/2}} \tag{2.2}$$

where

$$m_i = \frac{1}{N}\sum_{n=1}^{N}(x[n] - \bar{x})^i \tag{2.3}$$

2.3.3 Ground-truth acquisition

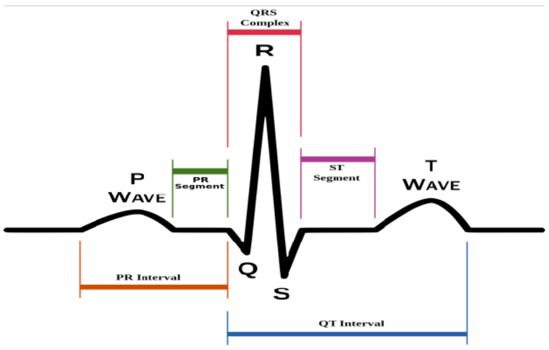
To establish a ground-truth heart activity reference, an electrocardiogram (ECG) was used as the ground-truth signal. The ECG provides a direct and well-established measurement of the heart's electrical activity and therefore serves as an accurate temporal reference for identifying individual cardiac cycles.

An ECG records the electrical activity of the heart as it contracts and relaxes. Its characteristic waveform, illustrated in Figure 2.3a, consists of several distinct components. Among its characteristic waveform components, the R-peak corresponds to the moment when the heart's ventricles are electrically activated to contract and pump blood into the arteries. This electrical activation causes the heart to contract and results in the physical ejection of blood from the heart to the arteries [31].

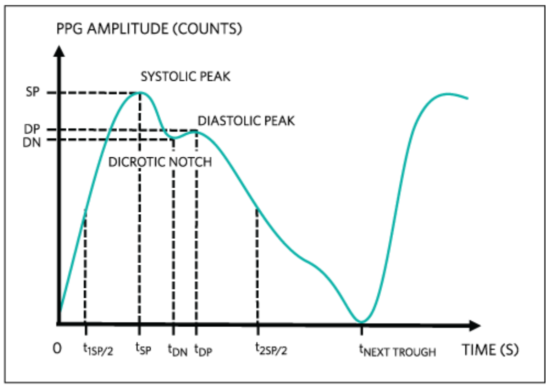
The ejected blood generates a pressure pulse wave that propagates through the arteries until it reaches peripheral sites such as the ear in this case. These waves can be optically captured by a PPG sensor, where the characteristic waveform of a PPG signal is shown in Figure 2.3b. When this pressure wave arrives at the measurement location, it causes a brief increase in local blood volume, which is observed as the systolic peak in the PPG waveform. Consequently, each PPG systolic peak originates from a cardiac contraction initiated at the corresponding ECG R-peak, but appears after a characteristic time delay. This delay, known as the pulse transit time (PTT), represents the time required for the pressure wave to travel from the heart to the peripheral measurement site [32].

By temporally aligning the ECG and PPG recordings, the ECG R-peaks provide a reliable reference for labelling the corresponding PPG (systolic) peaks [33]. This enables accurate mapping of the peripheral optical pulse to its underlying cardiac origin and allows for constructing a labelled ground-truth dataset.

To measure the ECG activity, a Byteflies ECG recording system [36] was used concurrently while the ear module was recording. This system was chosen as this was the only readily available ECG system. The Byteflies ECG recording system used was of model type 1.0.6, which comprised a Sensor Dot Model 1.0.1 and a dock of model 1.0.1. The ECG electrodes were placed in a single-lead configuration. One electrode was placed in the left parasternal region, approximately at the 3rd/4th



(a) ECG waveform (from [34])



(b) PPG waveform (from [35])

Figure 2.3: Annotated characteristic waveforms

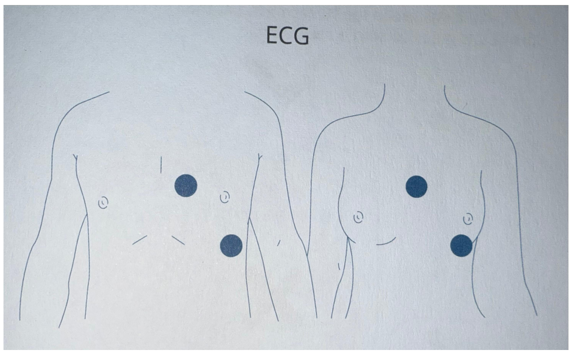


Figure 2.4: ECG electrode sensor locations (from [37])

intercostal space. The second electrode was placed on the left lateral thoracic wall, inferior to the pectoralis major muscle, near the 6th/7th rib at the mid-axillary line. This placement was the recommended placement by the ECG recording system and resulted in an ECG recording with minimal distortions. Figure 2.4 showcases a visual representation of the placement. This ECG system operated with a set sampling frequency of 250 Hz.

2.3.4 Automatic signal alignment

As the PPG and ECG systems operated asynchronously, the corresponding recordings required temporal alignment. Both devices contained a tri-axial accelerometer, which was leveraged to determine the relative time offset between systems caused by differing start-up times. During data collection, the subject performed between 3 to 7 jumps to create distinct acceleration peaks visible in both recordings. To ensure comparability between sensors that might be oriented differently, the three acceleration axes (x,y,z) were combined into a multiple orientation-independent metrics:

1. **Euclidean acceleration:**

$$a_{\text{eucl}} = \sqrt{a_x^2 + a_y^2 + a_z^2} \tag{2.4}$$

2. **Sum of absolute accelerations:**

$$a_{\text{sum}} = |a_x| + |a_y| + |a_z| \tag{2.5}$$

3. **Jerk-based acceleration:**

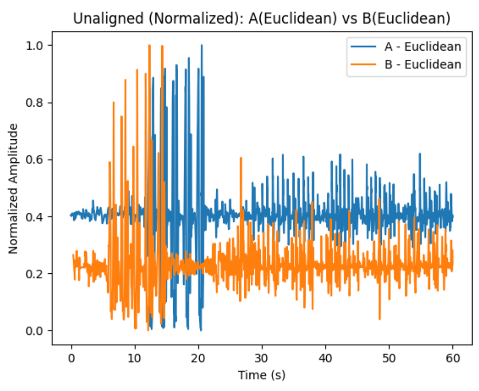
$$a_{\text{jerk}} = \sqrt{\left(\frac{da_x}{dt}\right)^2 + \left(\frac{da_y}{dt}\right)^2 + \left(\frac{da_z}{dt}\right)^2} \tag{2.6}$$

Each acceleration metric was normalised to the $[0, 1]$ range and cross-correlated between the two systems to identify the optimal temporal shift to align both acceleration metrics. The time offset Δt was determined as the lag corresponding to the maximum of the normalised cross-correlation function:

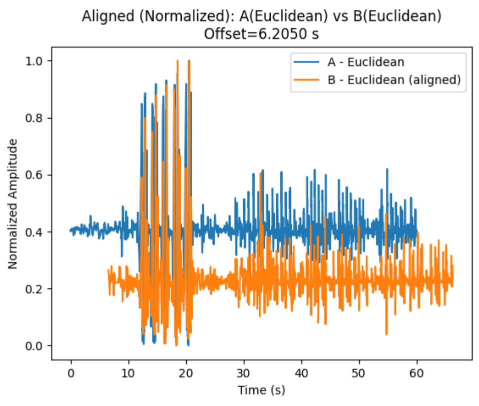
$$\Delta t = \arg\max_{\tau} \sum_{t} (a_1(t) - \bar{a}_1)(a_2(t+\tau) - \bar{a}_2) \tag{2.7}$$

where in Equation 2.7, $a_1(t)$ and $a_2(t)$ denote an acceleration metric magnitudes from the ear and chest acceleration sensors at time t respectively and $\bar{a}_1$ and $\bar{a}_2$ represent the mean values of each corresponding signal.

This process was fully automated to provide the relative offset between the PPG-system and the ECG-system efficiently. The resulting offset differences between different metric types typically fell within 0–50 ms, indicating that any metric could be used to achieve sufficient alignment accuracy. Minor errors introduced at this stage are corrected during the subsequent visual refinement step. In Figure 2.5, the results of the automatic alignment of the moving dataset of the shallow sensor placement in the external auditory meatus are outlined.



(a) Unaligned signals



(b) Aligned signal with offset=6.2050s

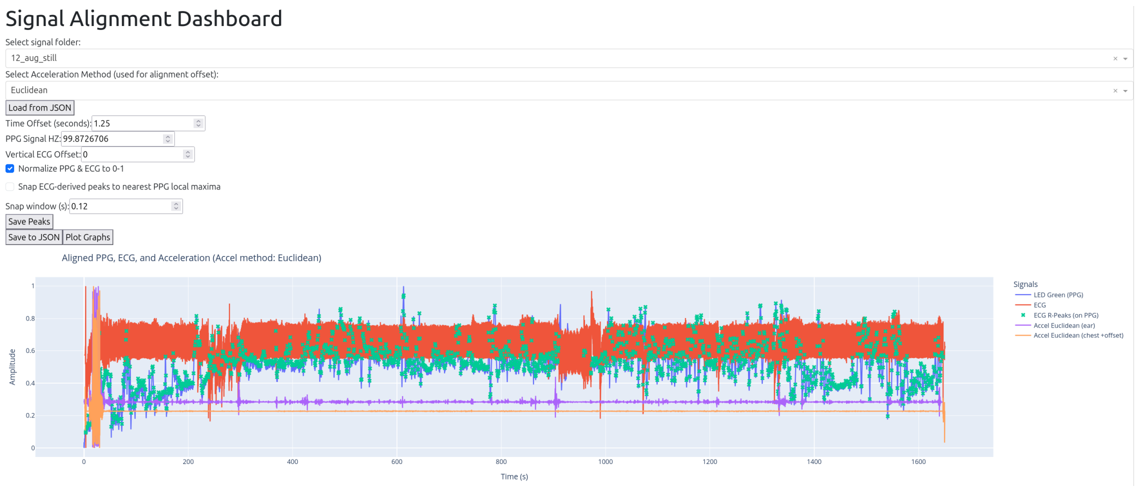
Figure 2.5: Automatic alignment of two acceleration signals by their normalised cross-correlation using the Euclidean metric (Equation 2.4)

2.3.5 Visual refinement and frequency calibration

After automatic general alignment, a custom web-based visualisation tool was used to fine-tune both the temporal offset and sampling frequency. The tool contains interactive controls for offset adjustment, normalisation, peak adjustment, and result saving. This web tool is depicted in Figure 2.6. Small deviations in the sampling frequency were corrected by comparing two reference peaks in both signals that experienced a consistent temporal drift over time. The derivation of the corrected sampling frequency is depicted in Equation 2.8.

$$f_{\text{new}} = f_{\text{old}} \times \frac{t_{\text{ref},2} - t_{\text{ref},1}}{t_{\text{meas},2} - t_{\text{meas},1}} \tag{2.8}$$

In Equation 2.8, $t_{\text{ref},1}$ and $t_{\text{ref},2}$ denote the expected time interval between two peaks in the reference ECG, and $t_{\text{meas},1}$ and $t_{\text{meas},2}$ represent the corresponding interval in the measured PPG. This correction ensured alignment consistency over long recordings where clock drift or sampling rate inaccuracies might otherwise accumulate.


Signal Alignment Dashboard

Select signal folder: 12_aug_still

Select Acceleration Method (used for alignment offset): Euclidean

Time Offset (seconds): 1.25

PPG Signal HZ: 99.8726706

Vertical ECG Offset: 0

[✓] Normalize PPG & ECG to 0-1

[ ] Snap ECG-derived peaks to nearest PPG local maxima

Snap window (s): 0.12

Figure 2.6: Custom web-based visual alignment tool for PPG & ECG signals

2.3.6 Signal validation and curation

The same visualisation interface was also used to inspect signal quality. Recordings exhibiting no identifiable PPG waveform were excluded from the final dataset. In cases where the lack of a usable PPG signal could be attributed to transient factors such as motion artefacts, sensor detachment, or external interference, an additional recording session was conducted to replace the corrupted one. However, if the signal degradation originated from an inherently poor sensor placement, such as an insufficient fit or limited optical coupling, no new recording was taken, as this reflected a realistic limitation of that particular configuration. These instances were documented and excluded from further analysis.

Similarly, corrupted ECG segments were identified and excluded from the corresponding data sources. These corrupted segments typically resulted in missing or distorted R-peak detection, which in turn led to incorrect or absent labeling of the associated PPG data. Only the affected portions of the ECG recordings were removed, while the remaining valid segments from the same session were retained for further analysis to preserve as much usable data as possible.

2.4 Configurable Deep Learning Framework for PPG Peak Detection

To optimise machine learning models for PPG peak detection, a specialised framework was developed to automate the complete pipeline, from data ingestion to embedded deployment. The framework was designed with a configurable architecture, allowing for the systematic comparison of configurations to identify optimal models, while ensuring adaptability for future applications.

2.4.1 Data Preprocessing and Annotation

The initial phase of the pipeline involves transforming raw sensor data into a format suitable for training neural networks. This process includes data discovery, signal standardisation, ground truth annotation using a reference ECG signal and data segmentation.

2.4.1.1 Data Ingestion and Organisation

The framework is designed to automatically detect data from multiple data sources and transform this into a single dataset. It operates on a structured top-level directory where each subdirectory represents a unique data source. Each data source is required to contain four files:

- `ppg.txt`: Raw packet output from the PPG measurement setup.
- `ecg.csv`: Reference ECG signal and its timestamps.
- `acc.csv`: Reference ECG system acceleration data.
- `data.json`: Metadata file containing necessary parameters about the data sets: PPG signal sampling frequency f_{PPG}, ECG signal sampling frequency f_{ECG}, temporal offset t_{off} and optionally excludable time windows.

2.4.1.2 Signal Standardisation and Resampling

Both ECG and PPG signals contain slightly varying sampling rates. To ensure a consistent temporal resolution across all data sources with correct peak timings, the framework implements a 1/2-stage resampling process on the raw PPG signal.

1. **Uniform Grid Alignment**: All PPG signals are resampled to a common target frequency fs_{target} (default 100 Hz) using polyphase filtering [38]. Given an input sampling rate fs_{in}, the rational factors are chosen as $p/q \approx fs_{\text{target}}/fs_{\text{in}}$. The resampled signal is defined as

$$y[m] = \sum_{k} x[k]\, h(mq - kp) \tag{2.9}$$

   where $h[\cdot]$ is a FIR low-pass filter to remove aliasing artefacts.

2. **Optional Downsampling**: To analyse the effects of different frequency rates lower than the original sampling rate, signals can be further resampled to fs_{final} using the same logic implemented in Equation 2.9 to achieve a new, lower frequency.

This standardisation ensures that all subsequent operations are performed on a consistent time base across the multiple data sources.

2.4.1.3 Ground Truth Generation via ECG Alignment

The core task is framed as a binary segmentation problem: classifying each time point in the PPG signal as either a "peak" (class 1) or "non-peak" (class 0). To generate accurate labels for the PPG signal, the synchronised ECG signal is used as a reference standard to create ground-truth labels. The process of using the synchronised ECG signal to construct ground-truth labels for the PPG signal involves multiple steps:

1. **R-Peak Detection**: The ECG signal must first be cleaned of power-line harmonics to ensure reliable R-peak extraction. Given the 250 Hz sampling rate, only interference components below the Nyquist frequency of 125 Hz can appear in the digitised signal. Consequently, the 50 Hz fundamental and its first harmonic at 100 Hz need to be removed.

   To suppress these components, two cascaded 2nd-order IIR notch filters with a quality factor of $Q = 30$ are applied at 50 Hz and 100 Hz. The quality factor relates the notch's centre frequency ω_0 to its -3 dB bandwidth bw as shown in Equation 2.10.

$$Q = \frac{\omega_0}{\text{bw}} \tag{2.10}$$

   The notch filters were applied using a forward–backward zero-phase filtering approach, in which the signal is filtered once in the forward direction and then again in reverse. This cancels the phase response of the filter, ensuring that no phase distortion or temporal shifting of the R-peaks occurs. The impact of the notch filtering is visible in a frequency domain representation shown in Figure 2.7, where the 50 Hz and 100 Hz power-line components are effectively attenuated. The corresponding time-domain signals are provided in Figure C.2 in the Appendix for completeness. The frequency response of the cascaded notch filters is shown in Figure C.1.

   After filtering, the ECG is normalised using Equation 2.11 to stabilise the subsequent processing stages.

$$\hat{x}_i = \frac{x_i - \min(x)}{\max(x) - \min(x)}, \quad i = 1, \ldots, N \tag{2.11}$$

   Finally, the XQRS algorithm [39] is applied to the cleaned ECG signal to identify the exact time indices of the R-peaks.

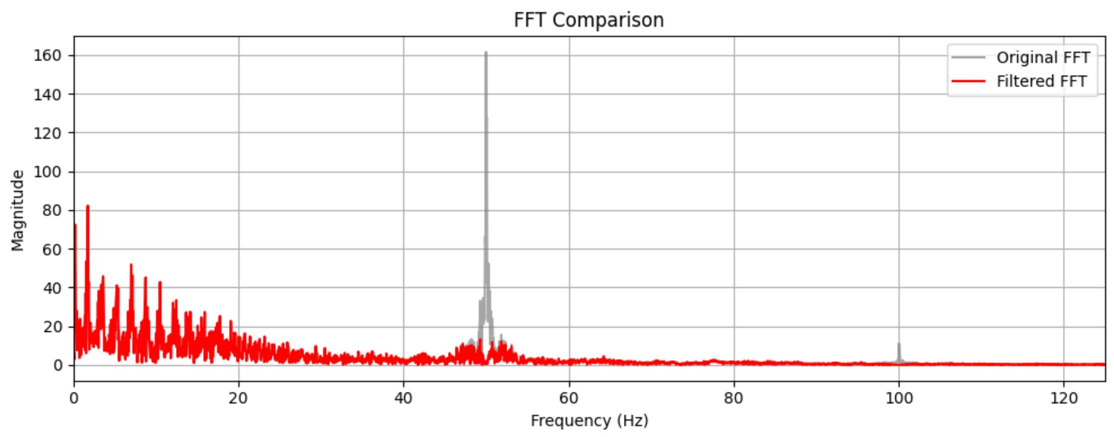


Figure 2.7: Frequency-domain comparison of the ECG signal before and after 50Hz and 100Hz notch filtering.

2. **Temporal Mapping**: The timestamps of the detected R-peaks are mapped onto the PPG signal's timeline by applying the earlier determined temporal offset specified in the metadata.

3. **Peak Adjustment**: As an optional refinement step, the peak indexes in the PPG signal are adjusted to ensure that each identified ECG R-peak correctly corresponds to its matching PPG peak. This step does not serve to reduce noise but rather to compensate for temporal mismatches between ECG and PPG peaks, primarily caused by minor differences in sampling frequencies and timing offsets in the experimental measurement setup.

   For each mapped R-peak time, the sample with the highest amplitude within a small symmetric local window of the PPG signal is selected. In a perfect scenario, this ensures that the ground truth label corresponds precisely to the local maximum of the PPG pulse wave, rather than the initial ECG-derived time point. In theory, the width of this window could be derived from the physiological limits of the human heart rate by using the relationship for the maximal heart rate per minute in healthy adults [40], which is shown in 2.12.

$$HR_{\max} = 208 - 0.7 \times \text{age} \tag{2.12}$$

   Dividing a minute by the absolute maximum value from this formula would correspond to a minimal inter-beat interval of $60.0 \text{ s} * 10^3/208 \approx 288$ ms. Using this value as the window range is however not advisable in practice, as in practice signal noise and motion artefacts may introduce local maxima with amplitudes larger than the PPG peaks present in the signal window. Instead, a window size of 20 ms was used. This value was empirically derived from recordings around 60 BPM (1 s per cardiac cycle). While effective for this dataset, the choice remains empirical and warrants further validation across different conditions.

   For each mapped R-peak time t_i, the sample with the highest amplitude within a local window $\mathcal{W}_i = [t_i - \Delta,\, t_i + \Delta]$ of the PPG signal is selected as shown in

Equation 2.13.

$$\hat{t}_i = \arg \max_{t \in \mathcal{W}_i \cap \{t_k\}} s(t), \tag{2.13}$$

In Equation 2.13, $s(t)$ denotes the PPG amplitude at time t.

4. **Label Caching**: To speed up subsequent experiments, the framework can cache these final refined peak locations in a specific optional file, bypassing the need for re-computation and ensuring consistency.

2.4.1.4 Windowing and Normalisation

The continuous annotated PPG signals are segmented into fixed-length windows with a pre-defined time step, the stride, between consecutive windows to provide input samples for the machine learning models. For each signal window, a corresponding binary label mask of the same length is generated, with a value of 1 at the locations of the peaks and 0 elsewhere. This windowing approach reflects how data would be received in a real-time scenario, where the model processes short segments of the incoming signal to detect peaks as they occur. A window length of 4 seconds with a stride of 1 second was selected based on preliminary evaluations, which showed a decent trade-off between sufficient temporal context and computational efficiency. Although the developed framework allows the window size and stride to be adjusted, a detailed analysis of its influence lies outside the scope of this research.

Furthermore, each window is independently min-max normalised to the range $[0, 1]$ following Equation 2.11. This local normalisation makes the model robust to variations in signal amplitude across different recordings or over time due to factors like sensor placement and subject movement.

2.4.2 Model Architectures

Two different CNN architectures were selected to introduce diversity in the model design and to assess how architectural choices influence performance in a temporally periodic task such as PPG peak detection. By including one model with standard convolutional layers and another incorporating dilated convolutions, the framework enables a more detailed investigation into how an increased receptive field affects the detection of periodic features. Since heart rate varies only gradually under normal physiological conditions, an architecture capable of integrating information over a longer temporal context may better capture the underlying periodic structure of the signal and improve the performance of its predictions.

2.4.2.1 Dynamic Encoder-Decoder CNN

The first model family is a standard, dynamically constructed encoder-decoder CNN. The network consists of an encoder, a decoder, and an output projection.

The architecture can be dynamically configured, allowing for systematic exploration of hyperparameters and different model configurations. Let the input signal be

$$x^{(0)} \in \mathbb{R}^{1\times T}, \tag{2.14}$$

where T denotes the temporal length.

Encoder The encoder applies B successive convolutional blocks, where each block comprises a convolutional layer, an optional batch normalisation layer, a non-linear activation function and an optional pooling layer. For block $i \in \{1, \dots, B\}$ with input $x^{(i-1)} \in \mathbb{R}^{C_{i-1}\times T_{i-1}}$, the convolutional layer in the convolutional block computes an output feature map z as in Equation 2.15.

$$z^{(i)} = W^{(i)} * x^{(i-1)} + b^{(i)}, \quad z^{(i)} \in \mathbb{R}^{C_i \times T_i}, \tag{2.15}$$

In Equation 2.15, $W^{(i)} \in \mathbb{R}^{C_i\times C_{i-1}\times k}$ are the convolutional kernels of size k, $b^{(i)} \in \mathbb{R}^{C_i}$ are the bias terms, and C_i denotes the number of output channels of the layer. In this context, each channel acts as an independent filter with its own randomly initialized weights. During training, these weights are iteratively updated through gradient descent, allowing each channel to ideally capture a distinct temporal or morphological pattern that contributes to an optimal prediction of the target output.

However, since dilation also play a factor, Equation 2.15 turns into Equation 2.16 in its discrete form with dilation implemented. Note that zero padding has been implemented to correct the input size for the specified kernel size and dilation.

$$z^{(i)}_{c,t} = \sum_{c'=1}^{C_{i-1}} \sum_{m=0}^{k-1} W^{(i)}_{c,c',m}\, x^{(i-1)}_{c',\,t+d\cdot m} + b^{(i)}_c, \quad z^{(i)} \in \mathbb{R}^{C_i\times T_i}, \tag{2.16}$$

where in Equation 2.16, the output is calculated for each channel c, with kernel size $k \in \mathbb{Z}_{>0}$ and dilation factor $d \in \mathbb{Z}_{>0}$. Note that this expression uses the cross-correlation operator rather than the mathematically flipped convolution. This follows modern implementations, which implement cross-correlation for efficiency to avoid filter reversal during training when using convolution [41].

After the convolutional layer, an optional batch normalisation [42] stage is potentially applied for improved learning rates. Theoretically speaking, by reducing internal covariate shift in the output of different layers, batch normalisation could stabilise training, enable higher learning rates, and promote more consistent feature learning across the PPG sequences in the dataset. Batch normalisation is implemented with Equation 2.17 (from [43]).

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta \tag{2.17}$$

where x denotes the input to batch normalisation (e.g. the output of a convolutional layer), $\mathrm{E}[x]$ and $\mathrm{Var}[x]$ are the mean and variance computed per channel over the mini-batch, ϵ is a small constant added for numerical stability to avoid division by zero, and γ and β are learnable scale and shift parameters, respectively.

Activation layer After the optional Batch Normalisation stage, an activation layer is applied to introduce non-linearity. Two activation functions were considered:

1. **SiLU (Sigmoid Linear Unit):** The SiLU activation function is defined as

$$\mathrm{SiLU}(x) = x \cdot \sigma(x) \tag{2.18}$$

where x is the output of the previous layer and $\sigma(x)$ is the logistic sigmoid function from Equation 2.19.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.19}$$

2. **ReLU (Rectified Linear Unit):** The ReLU activation function is defined as

$$\mathrm{ReLU}(x) = \max(0, x) \tag{2.20}$$

The ReLU activation zeroes out all negative values while keeping positive values unchanged.

Pooling Layers After a predefined number of convolutional blocks, a pooling layer is optionally applied to reduce the temporal dimension of the feature maps. Pooling operations aggregate local activations within a fixed or adaptive window, thereby improving translation invariance and reducing computational complexity in subsequent layers [44]. In this research, only fixed window pooling layers have been considered that could readily be implemented into the developed framework.

For a one-dimensional input feature map $x \in \mathbb{R}^{C \times T}$, pooling is applied independently per channel c over non-overlapping or adaptive windows of length p (in the code set to a constant of 2 for reduced complexity), producing an output $y \in \mathbb{R}^{C \times T'}$. The general formulation can be expressed as shown in Equation 2.21.

$$y_{c,t} = f\big(x_{c,\tau} \mid \tau \in [tp, (t+1)p)\big), \tag{2.21}$$

In Equation 2.21, $f(\cdot)$ denotes a local aggregation function. Depending on the pooling type, this function takes different forms:

1. **Max-Pooling:**

$$y_{c,t} = \max_{\tau \in [tp,(t+1)p)} x_{c,\tau} \tag{2.22}$$

Retains the largest value in the pooling window

2. **Average-Pooling:**

$$y_{c,t} = \frac{1}{p} \sum_{\tau \in [tp,(t+1)p)} x_{c,\tau} \tag{2.23}$$

Computes the mean value within each window

3. L_p**-Pooling:**

$$y_{c,t} = \left(\frac{1}{p} \sum_{\tau \in [tp,(t+1)p)} |x_{c,\tau}|^p \right)^{\frac{1}{p}} \tag{2.24}$$

A generalisation of Max- and Average-Pooling. For $p = 1$, it reduces to mean pooling; for $p \to \infty$, it approaches Max-Pooling. While many different values are possible for p, only $p = 2$ was realised in this research to include the L_p-pooling concept without shifting the primary focus of this research.

Decoder If the encoder reduces the temporal resolution of the feature maps through pooling operations, a decoder stage restores the original temporal scale to enable correct per sample predictions for peak detection. The decoder performs a simple nearest-neighbour upsampling operation with a total scale factor s of $s = 2^{N_p}$, where N_p is the number of preceding pooling layers. Mathematically, for a one-dimensional feature map $x \in \mathbb{R}^{C\times T}$ and upsampling factor s, the upsampled output $y \in \mathbb{R}^{C\times sT}$ is given by Equation 2.25.

$$y_{c,t} = x_{c,\lfloor t/s \rfloor} \tag{2.25}$$

In Equation 2.25, $\lfloor\cdot\rfloor$ denotes the floor operation. At last, a final convolutional layer transforms the upsampled multi-channel representation to a single output channel. This output channel contains the raw predictions of the model, commonly referred to as the model logits.

Network construction overview A high-level overview of the dynamic encoder-decoder CNN is shown in Figure 2.8, with the complete construction logic provided in pseudocode in Algorithm 1. This pseudocode summarises the structural logic, including the progressive channel expansion, pooling, and decoder upsampling.

2.4.2.2 Dilated Encoder-Decoder CNN

A second model family was developed to extend the encoder-decoder design. This model family first compresses the temporal resolution of the signal, after which convolution blocks with exponentially increasing dilation in the convolution layer are implemented. These convolution blocks (with the exponentially increasing dilation factor) combined will be referred to as the bottleneck stage, the stage where the network processes the features in their most compressed form, hence the term bottleneck. This architecture is more commonly known as a U-net architecture [45]. Concretely, the network comprises an encoder with two convolutional blocks, each containing a pooling layer, a bottleneck stage, and two convolutional blocks, each containing a nearest-neighbour upsampling layer instead of pooling layer. At last, an output projection is performed to obtain a single-channel output.

Encoder The encoder consists of two sequential stages, each performing downsampling. Each stage applies a 1D convolution (Equation 2.16), batch normalisation (Equation 2.17) and an activation layer (Equation 2.18 or Equation 2.20), followed by a pooling layer (Equation 2.22 to Equation 2.24). This encoder stage downsamples the signal to be used by the later bottleneck stage, which will perform most of the computations and is the core of this architecture.

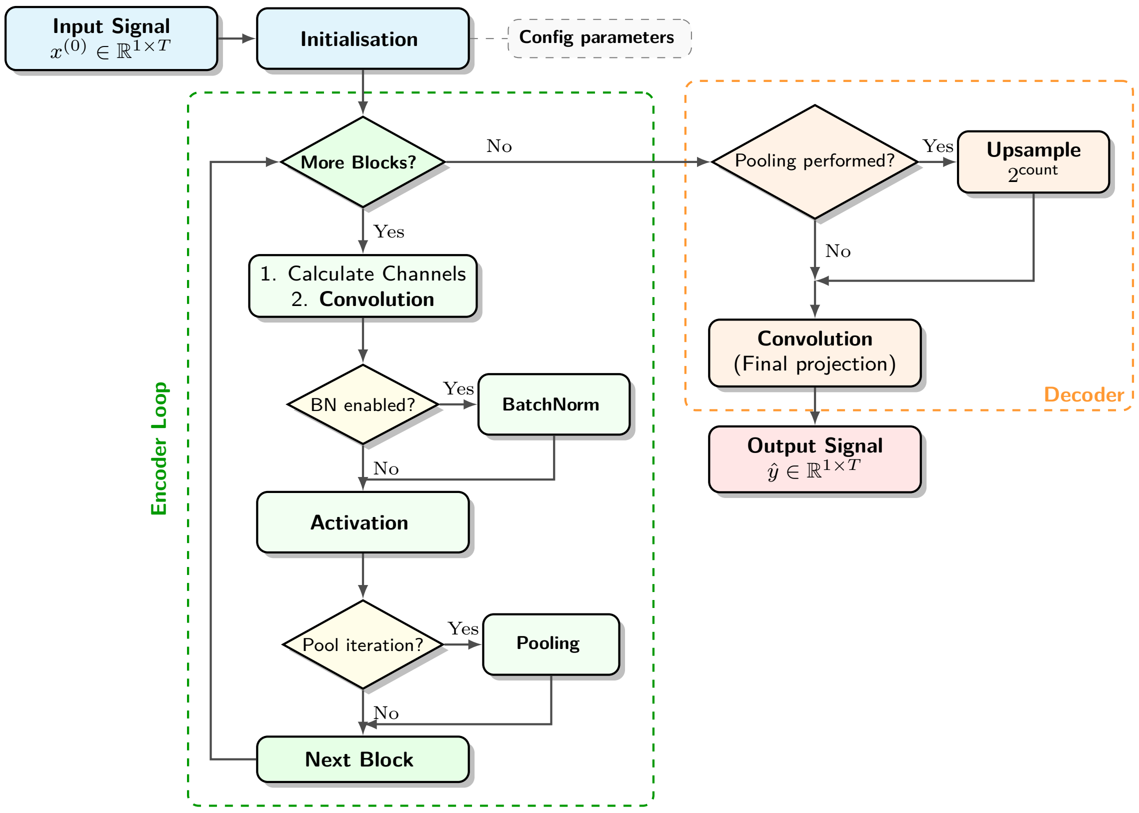


Figure 2.8: Architecture schematic of dynamic encoder-decoder CNN as outlined in Algorithm 1

Bottleneck with Dilated Convolutions After the initial downsampling, a bottleneck stage is introduced where the convolutional blocks each increase their dilation factor exponentially. This design allows the convolutional blocks to capture more signal context per convolutional layer, while keeping the number of parameters manageable for an embedded model. For the j-th convolutional block in the bottleneck, where $j \in \{0, \ldots, N_d - 1\}$ and N_d is the number of convolutional blocks in the bottleneck stage, the dilation factor d_j of each convolutional layer inside the block is given by Equation 2.26.

$$d_j = d_{\text{base}}^{j}, \tag{2.26}$$

In Equation 2.26, d_{base} is a user-defined base dilation factor (e.g. $d_{\text{base}} = 2$). This exponential growth of the receptive field enables the model to efficiently integrate information over a bigger time frame than would be possible with the aforementioned encoder-decoder CNN architecture (subsubsection 2.4.2.1).

Decoder The decoder reconstructs the original temporal resolution through two nearest-neighbour upsampling stages (Equation 2.25), each followed by a convolutional block to refine the enlarged feature maps. This interleaving of upsampling and convolution enables the network to regain finer temporal structure after the dilated bottleneck. After the final upsampling and convolution stage, a concluding convolution is applied to project the multi-channel feature maps down to a single-channel output signal $\hat{y} \in \mathbb{R}^{1 \times T}$.

The decoder stage symmetrically mirrors the encoder to restore the original temporal resolution of the input signal. It consists of two upsampling stages. Unlike the encoder-decoder CNN architecture's single upsampling step, this decoder interleaves nearest-neighbour upsampling (Equation 2.25) with a 'ConvBlock'. This approach allows the model to refine the upsampled features and learn to reconstruct finer details of the signal, which could help in producing a more precise peak-likelihood output. After the final upsampling and convolution stage, a concluding convolutional layer is implemented to obtain a single-channel output signal ($\hat{y} \in \mathbb{R}^{1 \times T}$.) from the multi-channel feature maps.

Network construction overview A high-level overview of the dilated encoder-decoder CNN is shown in Figure 2.9, with the pseudocode provided in Algorithm 2.

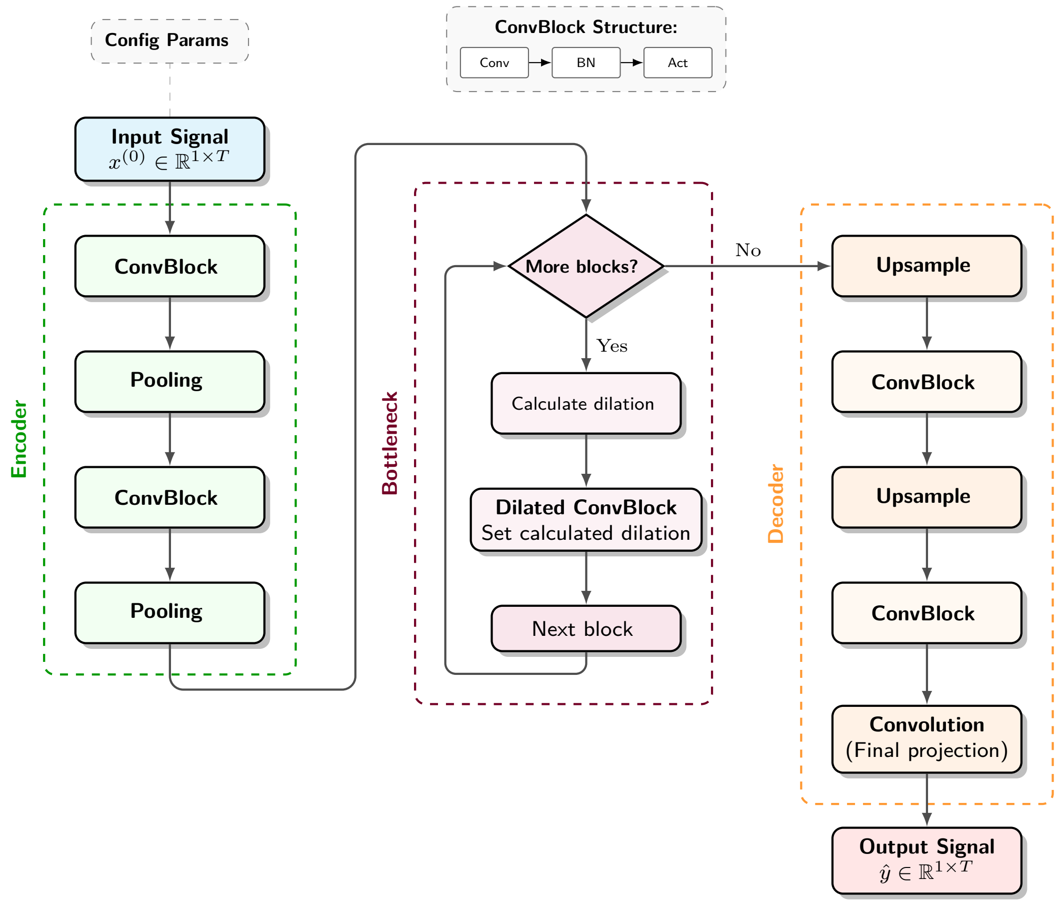


Figure 2.9: Architecture schematic of the Dilated Encoder-Decoder, as outlined in Algorithm 2
.

2.4.3 Model Training and Evaluation

The framework provides different options for training models and evaluating their performance using different metrics.

2.4.3.1 Training Procedure

For training the weights and biases, the AdamW optimiser is used [46]. Although alternative optimisation algorithms (e.g., SGD, RMSProp, or Adam) could have been considered, a comparative analysis of different optimisers falls outside the scope of this work and is therefore not explored further. Since the nature of determining PPG peaks leads to class imbalance between the R-peak and non-R-peak samples, with the severity of the class imbalance directly related to the sampling frequency, two different loss functions have been integrated in the framework for which the models aim to reach a minimum.

1. **Binary Cross Entropy**: Binary Cross Entropy (BCE) is the standard loss for binary classification tasks. Simply put, it penalises the difference between a predicted sample ($\hat{y}$) and the actual value (y). The equation can be seen in Equation 2.27 (from [47]).

$$l_n = -w_n \left[y_n \cdot \log \sigma(\hat{y}_n) + (1 - y_n) \cdot \log(1 - \sigma(\hat{y}_n)) \right], \tag{2.27}$$

   In Equation 2.27, l_n is the loss for a sample in the array, w_n is a weight sample, y_n is the actual sample value, $\hat{y}_n$ is the predicted sample value and σ is the sigmoid function as in Equation 2.19.

2. **Focal Loss**: An implementation of Focal Loss [48] is provided as an alternative to the standard BCE loss (Equation 2.27). By adding a modulating factor $(1 - p_t)^\gamma$, Focal Loss dynamically down-weights the loss contribution from easy, well-classified examples, thereby focusing the training process on harder-to-classify samples. The focal loss for a binary classification task can be defined as in Equation 2.28.

$$\begin{aligned} l_n &= -\alpha_n \, (1 - p_{t,n})^\gamma \log(p_{t,n}), \\ p_{t,n} &= \begin{cases} \sigma(\hat{y}_n), & y_n = 1, \\ 1 - \sigma(\hat{y}_n), & y_n = 0, \end{cases} \end{aligned} \tag{2.28}$$

   and $p \in [0, 1]$ is the model's estimated probability for the class with label $y = 1$. Here, $\alpha_t \in [0, 1]$ is a weighting factor that addresses class imbalance, and $\gamma \geq 0$ is the focusing parameter that controls the strength of down-weighting for easy examples. Setting $\gamma = 0$ reduces Focal Loss to standard BCE. Although implemented in the framework, only BCE is used in the experiments, as this proved to be the best-performing loss function in preliminary evaluations.

2.4.3.2 Post-Processing and Inference

The standard approach to obtaining binary predictions from a model's raw output (logits) is to apply a sigmoid activation function (Equation 2.19) and classify samples as positive when the resulting probability exceeds a predefined threshold. This straightforward method already produces valid binary outputs and is therefore used as the default inference procedure.

In addition to this standard method, two optional post-processing steps have been implemented to further refine the model's predictions:

1. **Thresholding**: Logits are first passed through a sigmoid function (Equation 2.19) to obtain probabilities. A tunable decision threshold is then determined to maximise the correct binary outputs. The framework automatically tunes this threshold on a validation set to maximise the MCC score (explained in Equation 2.30).

2. **Non-Maximum Suppression (NMS)**: To prevent the detection of multiple peaks for a single cardiac event, a 1D NMS algorithm is applied, more commonly used in 2D image detection [49]. This technique takes the list of predictions, selects the elements that have a predicted probability higher than a threshold τ and sorts the positive predicted samples by their probability prediction in descending order. Iteratively, the highest-scoring candidate is selected as a peak, and all neighbouring samples within a fixed suppression window are discarded. Equation 2.29 shows mathematically how a peak is selected. This process continues until no candidates remain which have not been selected or suppressed.

$$m[i] = \begin{cases} 1, & s[i] \geq \tau \text{ and } s[i] = \max\{s[j] \mid j \in [i-w, i+w]\}, \\ 0, & \text{otherwise.} \end{cases} \tag{2.29}$$

Although any window size can theoretically be chosen, in practice it should be selected in accordance with the temporal interval corresponding to the maximum expected heart rate (Equation 2.12).

2.4.3.3 Evaluation Metrics

The performance of the model is assessed using two complementary sets of metrics.

- **Sample-Level Metrics**: These metrics evaluate the model's performance at the level of individual time samples and include standard classification metrics such as precision, recall, macro F1-score, and the Matthews Correlation Coefficient (MCC). As [50] demonstrated that the MCC score provides a more reliable and informative measure for binary classification than accuracy or F1-score, MCC was selected as the primary metric to evaluate model performances. The MCC is defined as shown in Equation 2.30:

$$\text{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}, \tag{2.30}$$

  where TP, TN, FP, and FN denote the number of true positives, true negatives, false positives, and false negatives, respectively. The result of the MCC calculation is in the range $[-1, 1]$, with -1 implying complete misclassification and 1 implying perfect classification. An MCC value of 0 corresponds to random classification.

- **Event-Level Metrics**:

  Strict sample-level metrics could sometimes be too restrictive for certain applications. A more forgiving evaluation is achieved by assessing peaks within a configurable temporal window. In this metric, true and predicted peaks are paired together in a way that minimises the total matching error, essentially finding the most reasonable mapping between true and predicted peaks. This is achieved by using a linear sum assignment solver [51], with the underlying algorithm being a modified Jonker-Volgenant algorithm [52]. This event-based approach provides more flexibility to minor temporal deviations in the predicted peaks, without immediately classifying them as false positives. Although event-level metrics are implemented in the developed framework, they are not used in the experiments and for the reported results to ensure consistency and objectivity across model comparisons. However, their inclusion allows future studies to perform event-based evaluation when desired.

2.4.4 Hyperparameter Optimization Strategies

A key feature of the framework is its ability to perform automated hyperparameter searches to identify optimal configurations.

2.4.4.1 Grid Search

The framework can execute a grid search over a predefined hyperparameter space for the defined model architectures. It systematically trains and evaluates a model for each combination of parameters (e.g. number of layers, kernel size, specific pooling layer). The configuration resulting in the best performance on the validation set, determined by the MCC score, is identified as the optimal model.

2.4.4.2 Pareto Front Analysis

For a more complete understanding of model trade-offs, the framework implements a Pareto front search. This multi-objective optimisation approach seeks to find a set of models that represent the best possible compromise between different objectives. The objectives for this analysis are:

1. MCC Score: To be maximised.

2. Model Size (in Kilobytes): To be minimised.

The result of this search is a Pareto front [53], a set of non-dominated models where no single model can be improved in one objective without degrading its performance in another. This allows for an informed selection of a model that best fits specific deployment constraints, namely prediction accuracy with model size. The discovered optimal models, along with their configurations and performance metrics, can be saved for further analysis.

2.4.4.3 Model Size Exclusion Criterion

To ensure fair and practically relevant comparisons, all models exceeding a size of 2 MB were excluded in the training process from further analysis. This limit reflects the memory constraints of the embedded platform targeted in this work, 2 MB is the maximum flash capacity available in the STM32H7 microcontroller used in this research (section 2.5) for embedded validation experiments. Although the framework allows this threshold to be adjusted, it was fixed to 2 MB for this study.

2.5 Embedded model deployment

Since this research aims to create a model that can perform PPG peak detection on a resource-constrained embedded device, the final phase of the methodology involves deploying the trained model onto an embedded device. The STM32H755ZIT6U MCU [54] integrated on the Nucleo-H755ZI-Q development board [55], shown in Figure 2.10, has been used to test the model's validity and performance metrics. This device was selected because its architecture is representative of embedded systems suitable for in-ear PPG applications, while still providing sufficient resources to support a wide range of tests. Although its form factor (20x20x1.4 mm) does not allow for direct integration into an in-ear sensing device, it offers an accessible and practical platform for comprehensive evaluation. This embedded validation process is divided into three stages: model conversion, on-device deployment and performance validation procedures.

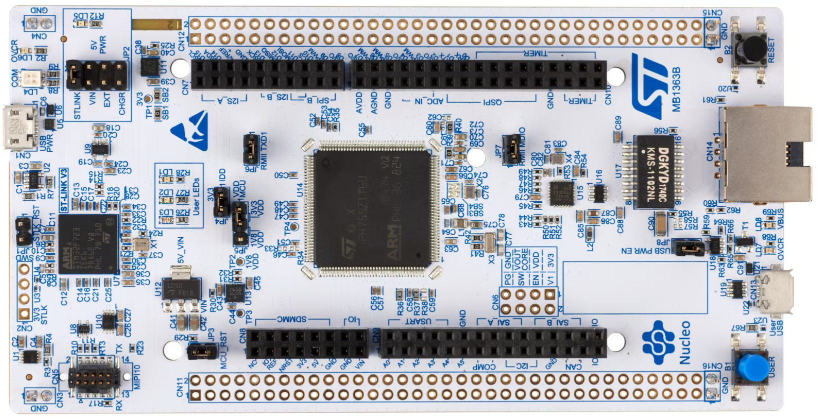


Figure 2.10: Nucleo-H755ZI-Q development board (from [55]).

2.5.1 Model conversion

As a first step, the developed models need to be converted to a format that is suitable for microcontrollers. LiteRT (or more commonly known as TFLite) [56] has been selected due to its widespread support and optimisation for edge devices. This toolset allows for conversion of different machine learning models to an embedded-optimised '.tflite' format [57], which can be used in later stages.

2.5.2 On-device deployment

The generated .tflite model is integrated into the STM32H7 firmware using the STM32Cube.AI toolchain provided by STMicroelectronics [58]. This software tool analyses the TFLite model and automatically generates an optimised C-code library for the STM32 architecture. This library includes functions to initialise the model, handle memory allocation for input/output tensors and activations, and execute the inference part.

STM32Cube.AI also includes an analysis mode that estimates the model's computational and memory requirements before deployment. It reports metrics such as the number of multiply–accumulate operations (MACs), Flash and RAM usage, and activation size for the selected embedded device. Although it does not provide exact runtime values, this analysis offers a reliable indication of model complexity and can be used to estimate the inference speed on the target device.

2.5.3 Performance validation procedures

To ensure that the conversion and deployment process did not introduce errors and to accurately measure on-device performance, a verification procedure was established. This verification procedure makes a direct comparison between the original developed ML model running on a PC and the converted embedded model running on the STM32H7.

The verification workflow is as follows:

1. **Prepare Input**: A standard PPG signal window is selected and pre-processed (min-max normalized, following Equation 2.11).
2. **PC Inference**: The script loads the original model and performs inference on the prepared input window. The raw output vector (logits) and the execution time in milliseconds are recorded.
3. **Embedded Inference**: The script transmits the normalised input window to the STM32H7 device over a serial (UART) connection.
4. **Device Execution and Reporting**: The STM32H7 firmware receives the data, runs a single inference using the integrated embedded model, and measures the exact number of CPU cycles consumed during the execution. It then transmits the resulting logit vector and the cycle count back to the PC.
5. **Comparative Analysis**: The script compares the logit vector from the PC against the one received from the microcontroller. To quantify the numerical equivalence, several metrics are computed:
   - **Mean Squared Error (MSE)** and **Root Mean Squared Error (RMSE)** measure the average squared deviation between predicted and

true outputs (Equation 2.31).

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \qquad \text{RMSE} = \sqrt{\text{MSE}}. \tag{2.31}$$

- **Mean Absolute Error (MAE)** quantifies the average absolute difference between the predicted and true values (Equation 2.32).

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|. \tag{2.32}$$

- **Cosine Similarity** evaluates the directional similarity between two output vectors in Equation 2.33, where a value of 1.0 indicates perfect alignment between the predicted and true vectors.

$$\text{Cosine Similarity}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\mathbf{y} \cdot \hat{\mathbf{y}}}{\|\mathbf{y}\| \, \|\hat{\mathbf{y}}\|} = \frac{\sum_{i=1}^{N} y_i \hat{y}_i}{\sqrt{\sum_{i=1}^{N} y_i^2} \sqrt{\sum_{i=1}^{N} \hat{y}_i^2}}, \tag{2.33}$$

This comparison validates that the embedded model is an (almost) exact representation of the original model before deployment on an embedded device. Furthermore, the reported CPU cycle count provides a precise and reliable benchmark of the model's computational cost on the embedded device, allowing for a direct assessment of its suitability for real-time applicability.

Alternatively, a full PC-based validation can be performed, following the evaluation methodology and metric definitions outlined by STMicroelectronics [59]. In this approach, both the original model and its equivalent embedded model are executed on a PC, allowing a direct numerical comparison between the two frameworks under identical software conditions, although not as applicable as the real embedded verification.

2.6 Experimental Design

Different experiments were performed to evaluate the developed in-ear sensing system and its corresponding machine learning models. Each experiment focused on a distinct stage of the pipeline: signal acquisition, model comparison, model configuration optimisation, and embedded deployment. Together, these experiments provide concrete results for assessing both sensing performance and model feasibility on embedded hardware from the developed framework.

2.6.1 Experiment 1: Sensor Signal Evaluation

The first experiment aimed to assess the quality and stability of the PPG signals obtained from the four in-ear sensor placements: tragus, concha, shallow external auditory meatus, and deep external auditory meatus. Using the dataset described in section 2.3, both seated and walking conditions were analysed to quantify the effect of motion on signal integrity.

The analysis consisted of two parts. First, a qualitative visual inspection was performed to evaluate the signal quality. Second, a quantitative assessment was conducted using the skewness factor (Equation 2.2) as an objective indicator of PPG waveform quality. A positive skewness value (> 0) was used as an indication for well-formed, physiologically plausible signals.

The results of this experiment established the baseline signal characteristics for each ear location and provided initial results on which placement offers the most reliable signal morphology for subsequent machine learning experiments. The results are presented in section 3.1, with the results interpreted and discussed in section 4.1.

2.6.2 Experiment 2: Comparative Model Analysis Across Sensor Locations

The second experiment evaluated how sensor placement affects the predictive performance of a representative subset of machine learning models trained for PPG peak detection. The different models were trained on their respective dataset ((section 2.3), with the models corresponding to the configuration space outlined in Table B.3 under the *Comparative* column. The results are presented in section 3.2, with the results interpreted and discussed in section 4.2.

2.6.3 Experiment 3: Model Configuration and Parameter Sensitivity

After identifying the most suitable sensor placement, a detailed analysis was conducted on a single location. The deep external auditory meatus was selected as this proved to be the best performing placement from the previous experiment. This experiment aimed to assess how individual model configuration parameters influence predictive performance and model size.

To ensure generalisable insights, the configuration space was designed to vary not only the parameter under investigation but also other parameters within representative bounds. This approach prevents the analysis from depending on a single fixed model setup and instead captures the averaged effect of each parameter across a broad range of configurations. The complete parameter spaces used for this analysis are outlined in Table B.1 and Table B.2, corresponding to the standard and dilated CNN families, respectively. Models exceeding the 2 MB memory threshold were excluded from further analysis.

Both model families are evaluated, with the following sections describing each model family and the parameters examined in detail.

2.6.3.1 Dynamic Encoder–Decoder CNN.

The first part of the experiment focused on the dynamic encoder–decoder CNN architecture (Algorithm 1). The following configuration parameters were investigated individually:

- **Batch Normalisation**:
  Compared models trained with and without batch normalisation to quantify its effect.
- **Pooling frequency and type** (`pool_every`, f_{pool}):
  The downsampling interval was analysed with intervals of 1,2,4,8 and the impact of the pooling function used (max, average, L_p).
- **Base channel width and scaling factor** (`base_channels`, `double_every`):
  Analysed the effects of the initial number of channels (8, 16, 32 and 64) and the number of blocks after which the number of channels are doubled (1, 2, 4 and 8).
- **Kernel size and dilation factor** (`kernel_size`, `dilation`):
  Investigated the impact of increasing the kernel width (3, 5, 7 and 9) and spacing between kernel elements (1, 2, 4 and 8) in the convolution layer.
- **Activation function:** Compared the use of `ReLU` and `SiLU` as an activation layer.

The results are presented in section 3.3, with the results interpreted and discussed in section 4.3.

2.6.3.2 Dilated Encoder–Decoder CNN.

In the second part, the Dilated Encoder–Decoder CNN (Algorithm 2) was examined. Within this model family, the following configuration parameters were explored:

- **Number of dilated layers** (`num_dilated_layers`): The number of layers in the bottleneck stage (3, 5 and 7) were varied to evaluate their influence.
- **Bottleneck channel width** (`bottleneck_channels`): Analysed the impact of increasing the internal channel width (16, 32, 64 and 128).
- **Bottleneck stage kernel size and base dilation factor** (`kernel_size`, `base_dilation_dc`): Explored how the kernel size (3, 5, 7 and 9) and the base dilation factor (1, 2, 3 and 4) in the bottleneck stage influence the model characteristics.
- **Encoder channel and kernel size** (`enc1/enc2_channels`, `enc1/enc2_kernel`): Investigated how varying the number of channels and kernel widths in the encoder stages affects performance and model compactness.
- **Decoder similarity**: Evaluated whether maintaining parameter symmetry between encoder and decoder (matching channels and kernel sizes) improves performance, or whether independently configured encoder/decoder parameters are better.
  In this experiment, a symmetric design leads to the best performance (subsubsection 3.3.2.4). Therefore, no further analysis of the independent parameters are performed.

The results are presented in subsection 3.3.2, with the results interpreted and discussed in section 4.4.

In addition to the architectural parameters, this experiment also examined the influence of pre- and post-processing steps, namely peak refinement and non-maximum suppression (NMS). The corresponding configuration settings are listed in Table B.3, with the results presented in subsection 3.3.3, with the discussion taking place in section 4.5.

2.6.4 Experiment 4: Embedded Validation

The final experiment validated the developed embedded model, ensuring that the methodology resulted in numerically correct embedded models that were also executable within embedded hardware and timing constraints. Characteristic models from the previous experiments were converted to an embedded format and deployed on a STM32H755 microcontroller. In this experiment, the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Cosine Similarity were computed to ensure numerical equivalence (Equation 2.31-2.33).

Two complementary analyses were performed:

1. **Host/PC validation:** The embedded-format models were first executed on a PC to verify that the conversion process preserved numerical correctness.
2. **Embedded validation:** The same embedded-format models were then deployed on the STM32H755 microcontroller to evaluate their numerical behaviour on-device. Additionally, the execution time of each model was measured.

This final experiment ensured that the selected models were not only accurate but also executable within the hardware and timing constraints of an embedded in-ear sensing platform.

The results are presented in section 3.4 and discussed in section 4.6. The configuration space used is outlined in Table B.3 under the *Embedded* column.

2.7 Evaluation Metrics and Visualisations

The following visualisation types are used throughout the result chapter of this thesis (chapter 3) to summarise and compare model performance across configurations.

- **Box plots:** These plots summarise the distribution of MCC values by showing the median and the interquartile range (IQR). The whiskers extend to the most extreme data points that lie within 1.5 $\times$ IQR from the quartiles, indicating the range of typical values. Any data points beyond this limit are plotted individually as outliers. See Figure 3.4 as an example.
- **Pairwise win-rate matrices:** These heatmaps show, for every pair of parameter settings, the proportion of configurations in which one setting yields a higher MCC than the other. See Figure 3.5 as an example.

- **Histograms of ΔMCC:** These histograms present the distribution of MCC differences between paired configurations for binary parameter choices, such as models trained with or without peak adjustment. See Figure 3.6 as an example.
- **Scatter plots (MCC vs. model size):** These plots illustrate the relationship between MCC values and model size across different configurations. See Figure C.3 as an example.

Results 3

In this section, the results from the different experiments (section 2.6) are presented. First, the PPG measurements from all four in-ear sensor locations are shown, including qualitative waveform evaluations and skewness statistics. Second, the outcomes of the comparative analysis are reported, summarising model performance across sensor placements. Third, the detailed results of a single-location parameter study of both architecture families are shown, with the effects of pre- and post-processing steps. Last, the results of the embedded validation are reported. The results are solely presented in this section, the discussion of these results can be found in chapter 4.

3.1 PPG Sensor Results

In this section, the PPG measurements from the in-ear PPG sensor, placed in the four different locations (as outlined in subsection 2.1.3), will be visually inspected in both seated and walking condition.

3.1.1 Visual Observation

The visual analysis begins with the seated measurements, followed by the walking condition.

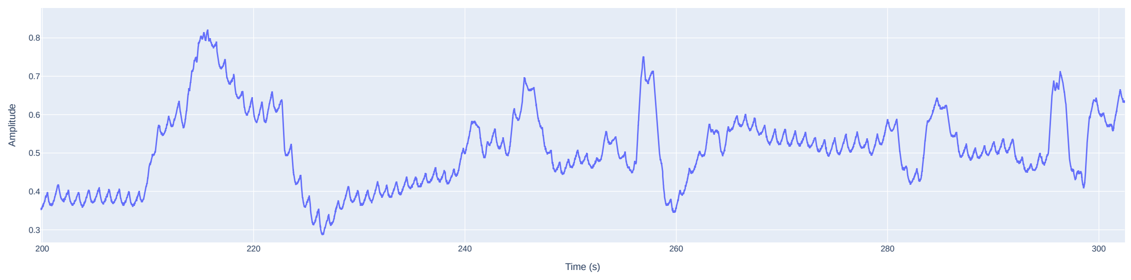
3.1.1.1 Seated condition

Figure 3.1 illustrates the PPG waveforms recorded during seated measurements across all sensor placements. Each graph shows a 100-second segment (200–300 s in the recording), normalised to the range $[0, 1]$ for comparability.

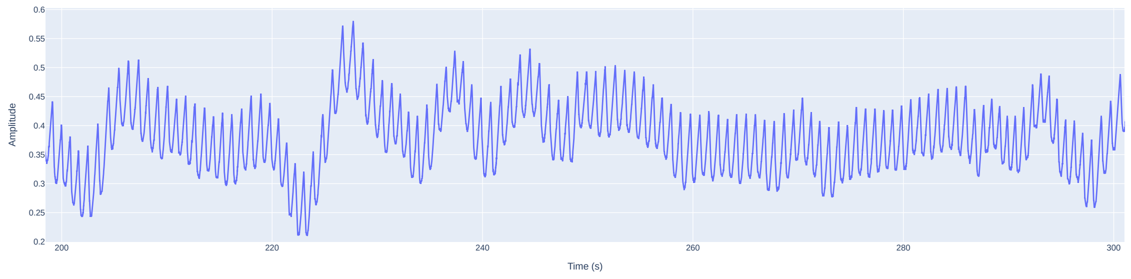
In all locations, a periodic, sinusoid-like pattern corresponding to the expected behaviour of PPG-signal, with approximate frequencies between approximately $1 - 1.2$ Hz ($\approx 60 - 70$ bpm) can be seen. Furthermore, the timing of these periodic fluctuations aligns with the peaks in the simultaneously recorded ECG. Moreover, small variations in waveform amplitude and baseline drift are visible between locations.

3.1.1.2 Walking condition

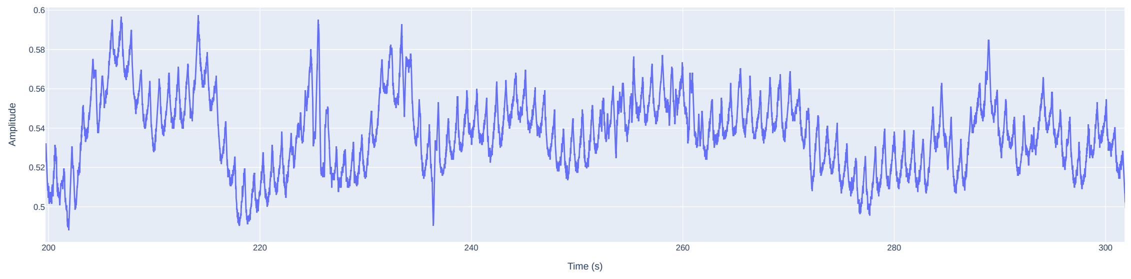
Figure 3.2 presents PPG segments acquired while the subject was walking. Under these conditions, motion artefacts become prominent, and the amount of signal distortion varies considerably between sensor placements. The concha signal (Figure 3.2b) contains a recognisable PPG morphology throughout most of the segment, although mild amplitude differences and baseline fluctuations are present. The deep



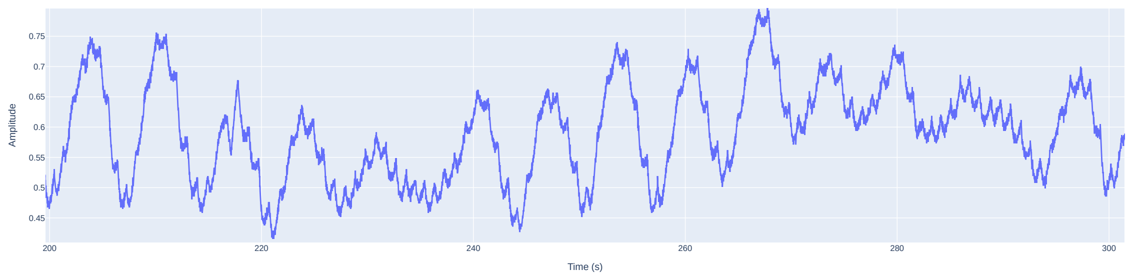
(a) Deep meatus



(b) Concha



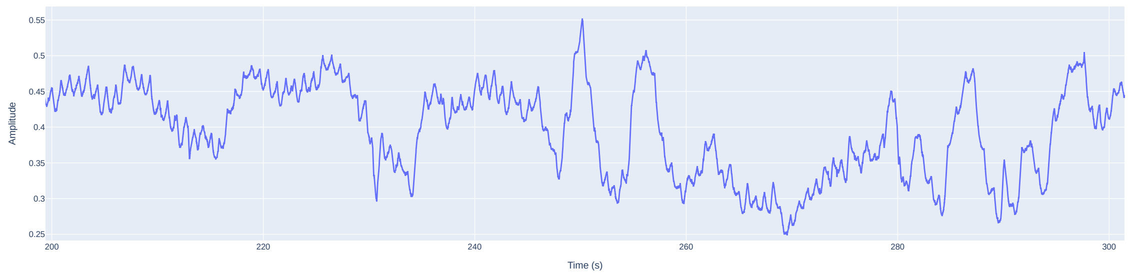
(c) Tragus



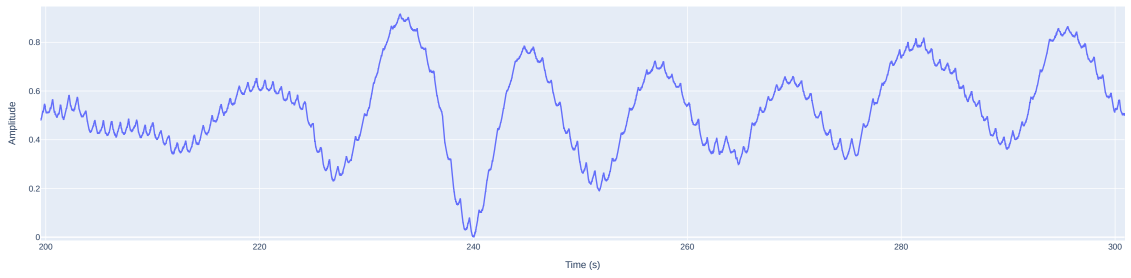
(d) Shallow meatus

Figure 3.1: Sample 100-second PPG signal segments (200–300 s) recorded under seated conditions across all sensor placements.

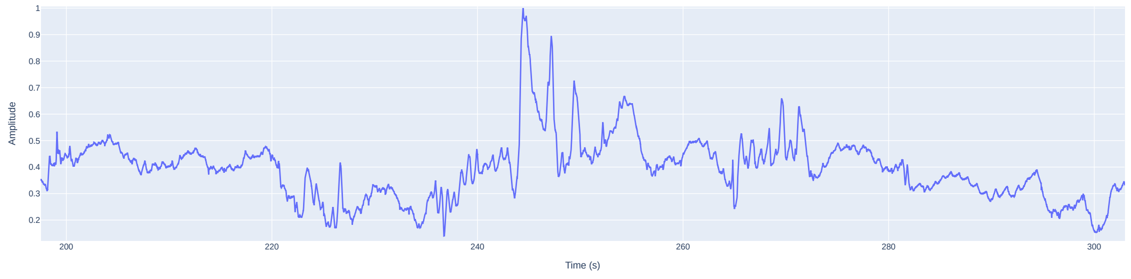
meatus signal (Figure 3.2a) shows more frequent distortions and irregular loss of periodicity, particularly between 240–260 s, yet partial waveform structure remains visible. In contrast, the tragus and shallow meatus signals (Figure 3.2c and Figure 3.2d, respectively) display severe artefacts that obscure the underlying cardiac rhythm, making individual pulses visually indistinguishable in several intervals (e.g. 240–270 s for the tragus). To illustrate this effect, ECG R-peaks were overlaid on the 240-260 s time frame in Figure 3.3, showing that the temporal correspondence between cardiac events and optical PPG peaks is largely lost for these locations.



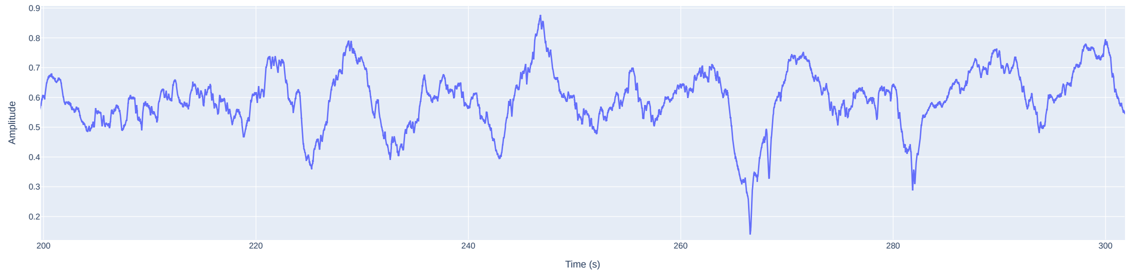
(a) Deep meatus



(b) Concha



(c) Tragus

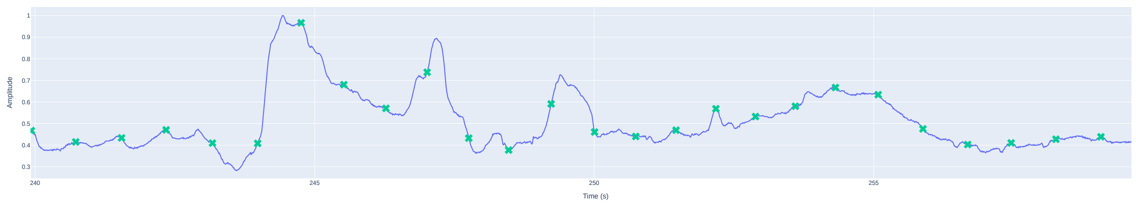


(d) Shallow meatus

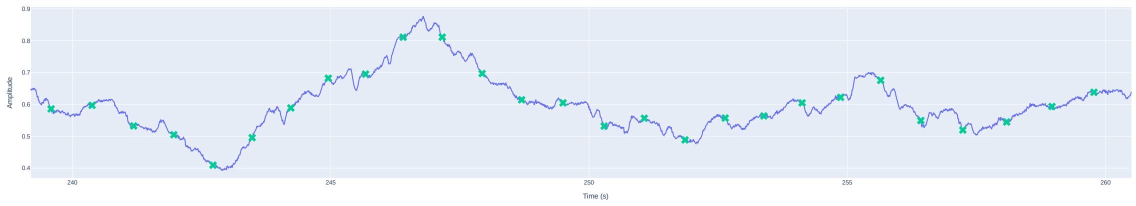
Figure 3.2: Sample 100-second PPG signal segments (200–300 s) recorded under walking conditions across all sensor placements.

3.1.2 Skewness

Table 3.1 summarises the skewness characteristics of the PPG signals for all evaluated sensor locations and activity conditions. For each recording, the median, mean, interquartile range (IQR), and standard deviation (SD) were computed across all analysis windows to quantify the distribution of skewness values.



(a) Tragus



(b) Shallow Meatus

Figure 3.3: Sample 20-second PPG signal segments (240–260 s) recorded under walking conditions for tragus and shallow external auditory ear meatus with ECG R-peaks indicated by green crosses.

		Skewness			
Location	**Condition**	**Median**	**Mean**	**IQR**	**SD**
External auditory meatus (deep)	Non-moving	0.092	0.111	0.591	0.539
External auditory meatus (deep)	Moving	0.099	0.123	0.640	0.557
External auditory meatus (shallow)	Non-moving	−0.030	−0.065	0.658	0.497
External auditory meatus (shallow)	Moving	−0.117	−0.140	0.725	0.563
Concha	Non-moving	0.260	0.201	0.334	0.311
Concha	Moving	−0.024	−0.047	0.924	0.598
Tragus	Non-moving	0.085	0.117	0.800	0.780
Tragus	Moving	0.065	0.073	0.800	0.620

Table 3.1: Summary of PPG signal skewness across sensor locations and activity conditions. Values represent the median, mean, interquartile range (IQR), and standard deviation (SD) computed over all analysis windows.

3.2 Comparative Analysis

Following the qualitative assessment of PPG signal quality, this subsection presents a quantitative comparison of the ear sensor locations. For the locations, the tragus, deep inside the external auditory meatus, shallow inside the external auditory meatus and the concha will be evaluated (as outlined in subsection 2.1.3). The evaluation will be performed based on the performance of machine learning models trained on the location-specific datasets. The goal is to determine how sensor placement affects model performance and identify the location(s) providing the most reliable cardiac signal for automated PPG peak detection. Performance was quantified using the Matthews Correlation Coefficient (MCC, Equation 2.30), which quantifies the accuracy of cardiac peak detection.

3.2.1 Results

Figure 3.4 presents the distribution of MCC scores across the four sensor locations. In total, 96 configurations were evaluated for each placement. A clear upward trend is noted in both the mean/median MCC and the maximum values when ordering locations from Tragus → External auditory meatus (Shallow) → Concha → External auditory meatus (Deep). The corresponding numerical results are provided in Table D.1.

The pairwise comparison matrix in Figure 3.5 supports these results, with the deep external auditory meatus placement achieving win rates, the percentage of configurations achieving a higher MCC than the other, of 92.2%, 100.0%, and 100.0% over the concha, shallow external auditory meatus, and tragus placement, respectively. The concha sensor location similarly outperforms the shallow external auditory meatus and tragus placements, with win rates of 98.9% and 100.0%, respectively.

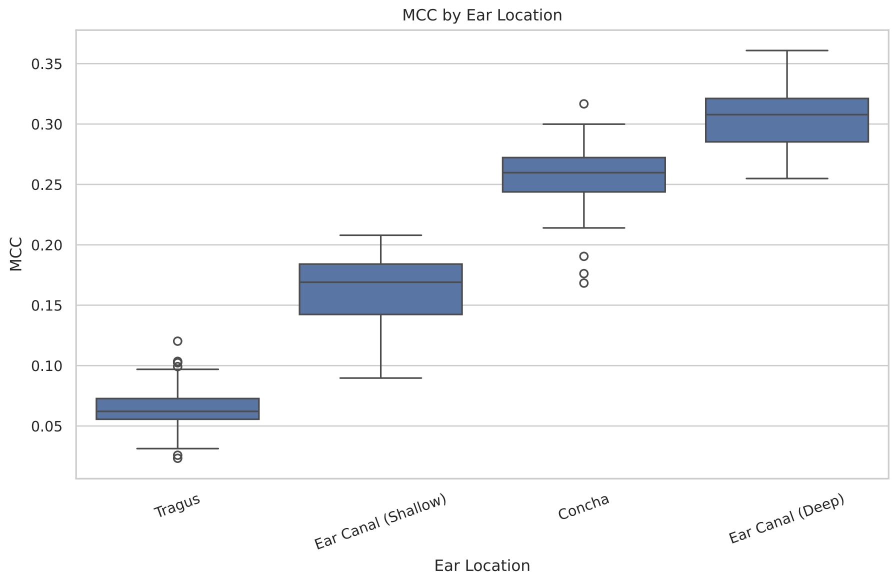


Figure 3.4: MCC box plots for different sensor locations

3.3 Single-location in-depth analysis

In this section, a detailed analysis is conducted for a single sensor location to evaluate the influence of different parameter settings for both model architecture families. The objective is to identify viable configurations and gain insights into the effect of different parameters. As a location, the deep external auditory meatus is used. While any different location could be chosen, this location proved to be the most viable location in earlier results.

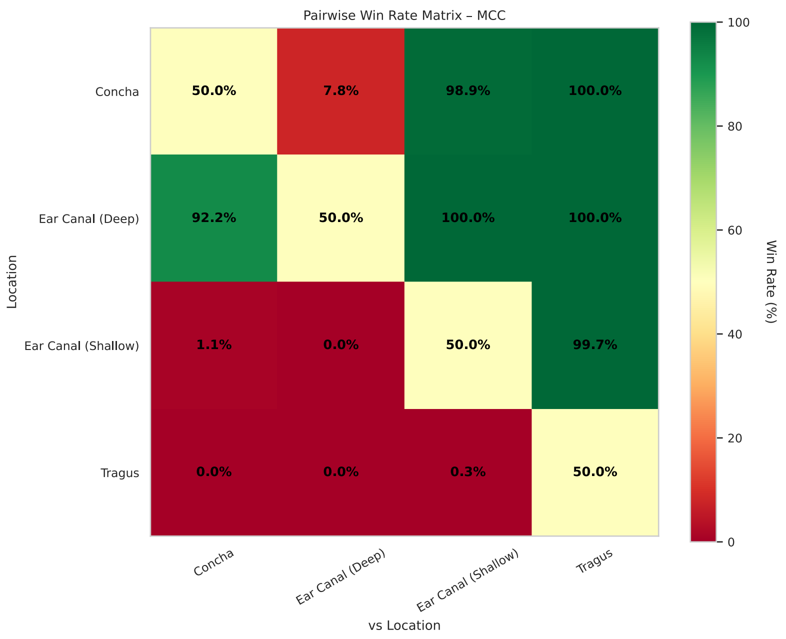


Figure 3.5: Heatmap for MCC win rates by ear location

3.3.1 Standard CNN

In this analysis, the following configuration parameters of the standard CNN are evaluated individually: batch normalisation, pooling frequency and type, base channel width and scaling factor, convolutional kernel size and dilation, and activation function (subsubsection 2.6.3.1). The subsequent subsections present the corresponding results, with an overview of the numerical results provided in Table D.2.

3.3.1.1 Batch Normalisation

Initially, the effect of batch normalisation (BN) on model performance is presented. The results, presented in Figure 3.6, demonstrate a consistent improvement in performance when BN was applied. Specifically, 30 configurations (93.75%) achieved a higher MCC with batch normalisation enabled, while only 2 configurations (6.25%) performed better without batch normalisation. Based on these findings, batch normalisation was included by default in all subsequent experiments to reduce the search space and focus computational resources on more influential architectural and training parameters.

3.3.1.2 Pooling strategies

Another important aspect concerns the pooling frequency and type, defined by the `pool_every` and pooling parameter in Algorithm 1 respectively. These parameters determine after how many convolutional blocks a pooling layer is applied and of what type.

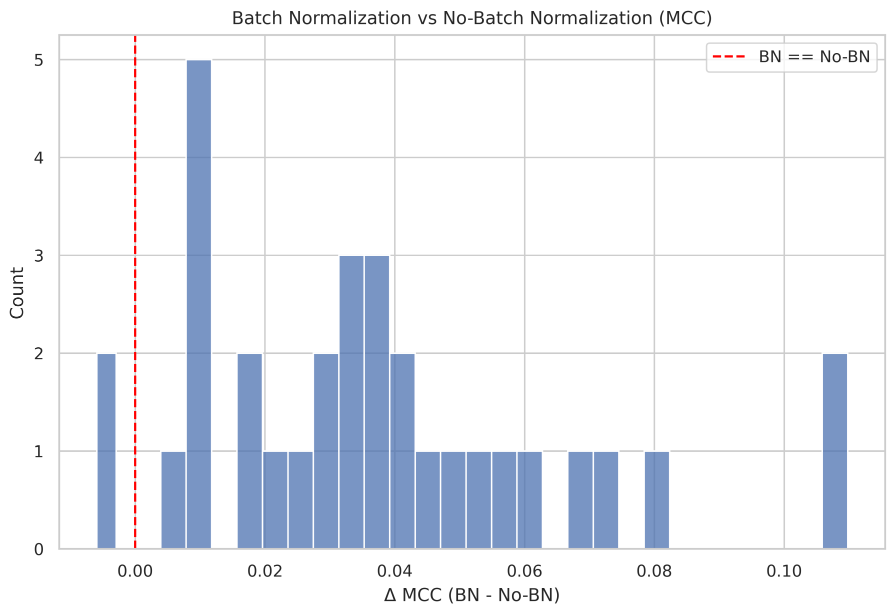


Figure 3.6: Distribution of the change in Matthews Correlation Coefficient (ΔMCC) between models trained with and without batch normalisation (BN). A total of 30 configurations (93.75%) achieved higher MCC with BN, while only 2 configurations (6.25%) performed better without BN.

Pool_every Figure 3.7 shows the distribution of MCC scores for different `pool_every` values.

As visible in the figure, increasing the pooling interval from 1 to 2 improved the mean and median MCC values, while further increases to 4 and 8 led to only minor variations. The total number of parameters and resulting model sizes remained consistent across configurations (mean/median of 67.5 kB/11.7 kB, range: 1.9–857.1 kB), given that all other parameters were held constant, which is to be expected.

Furthermore, Figure C.3 illustrates the relationship between `pool_every`, model size, and the resulting MCC across configurations. Complementarily, Figure C.4a presents a pairwise comparison matrix of `pool_every` values, where each cell indicates the proportion of configurations in which one setting achieved a higher MCC than another (the win rate).

Pooling type Figure 3.8 shows the distribution of MCC scores for the three evaluated pooling types (*max*, *avg*,*lp_2*).

As observed from the figure, all pooling methods resulted in comparable performance, with mean/median MCC values around 0.20/0.22. The MCC range across models remained consistent, spanning approximately from 0.02 to 0.28. The corresponding win-rate matrix can be observed in Figure C.4b.

3.3.1.3 Channels

Another aspect of the model is the number of channels and the increase thereof, set by the `base_channels` and `double_every` parameters in Algorithm 1.

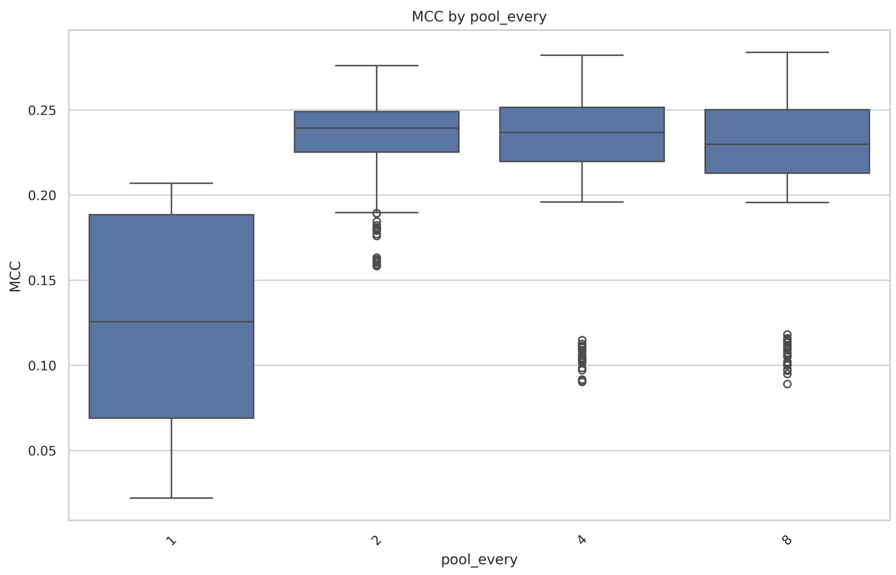


Figure 3.7: Box plot showing the distribution of MCC scores across 192 trained models for each `pool_every` configuration

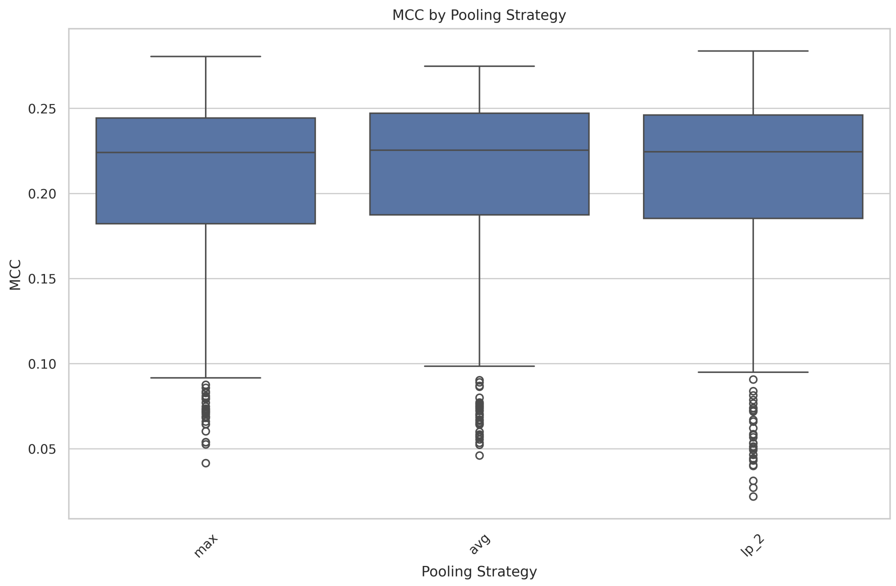


Figure 3.8: Box plot showing the distribution of MCC scores across 256 trained models for each different pooling layer

Base channels Figure 3.9 presents the distribution of MCC scores across different `base_channels` values. The mean/median MCC values showed a consistent upward trend with increasing channel width, ranging from 0.230/0.236 at 8 channels to 0.243/0.251 at 64 channels. Across all configurations, MCC scores spanned approximately from 0.100 to 0.292.

Model size increased substantially with higher `base_channels` values, ranging from a mean of 27.43 kB at 8 channels to 380.47 kB at 64 channels, with the corresponding box plot of model sizes shown in Figure C.7a. The pairwise comparison matrix in Figure C.6a confirms that higher `base_channels` settings achieved higher win rates, with the 64-channel configuration outperforming the 8, 16, and 32 configurations in 71.1%, 59.7%, and 55.7% of pairwise comparisons, respectively. Finally, Figure C.8 shows the relationship between model size and MCC scores for different `base_channels` values.

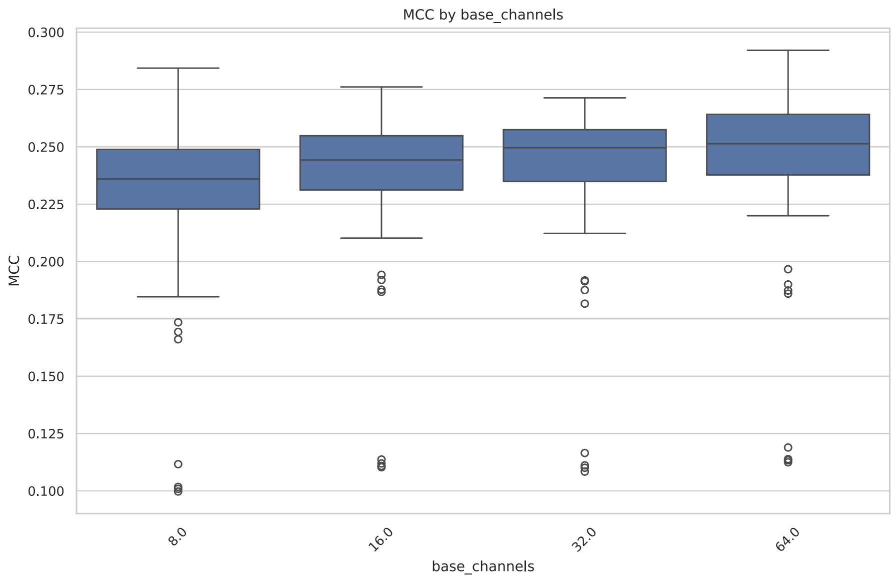


Figure 3.9: Box plot showing the distribution of MCC scores for each different `base_channel` value across 360 total configurations

Double every Figure 3.10 presents the distribution of MCC scores across different `double_every` values. The mean/median MCC values were similar across all settings, ranging from 0.236/0.244 at 8 to 0.239/0.249 at 2. Across all configurations, MCC scores spanned approximately from 0.10 to 0.29. As can be seen in Figure C.9, the maximum MCC value of 0.292 was obtained for a model of approximately 1.2 MB, representing the largest model within the dataset. For `double_every`=1, the best-performing model reached an MCC of 0.284 with a model size of about 200 kB.

Model sizes decreased systematically with increasing `double_every` values, ranging from a mean/median of 287.4 kB/167.4 kB at 1 to 65.7 kB/23.5 kB at 8, observable in Figure C.7b. The pairwise comparison matrix in Figure C.6b shows that lower `double_every` settings achieved slightly higher win rates, with the `double_every`=1 configuration outperforming the configuration with a value of 8 in 55.0% of pairwise comparisons and the configuration with a value of 2 outperforming the `double_every`=8 configuration in 55.9% of cases. Finally, Figure C.9 illustrates the relationship between model size and MCC scores across the different `double_every` values.

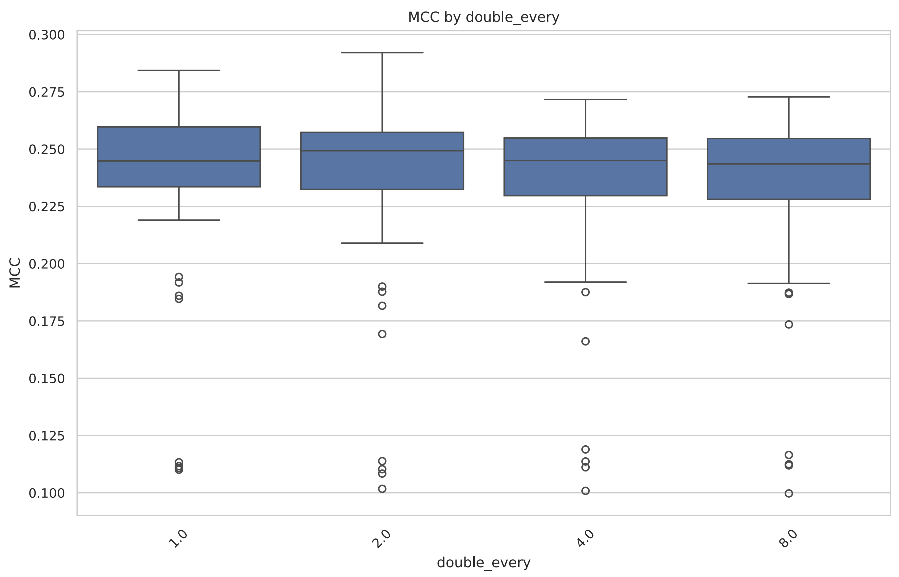


Figure 3.10: Box plot showing the distribution of MCC scores for each different `double_every` value across 360 total configurations

3.3.1.4 Convolution Layer

The convolution layer contains two important parameters, namely the kernel size and the dilation factor. The kernel size defines the width of the convolutional sliding window, while the dilation factor specifies how many zeros are inserted between adjacent kernel elements, effectively expanding the receptive field. The effect of both parameters will be presented in this subsection.

Kernel size Figure 3.11 presents the distribution of MCC scores across different `kernel_size` values. The mean/median MCC values increased slightly with kernel size, ranging from 0.224/0.233 at a kernel size of 3 to 0.255/0.256 at a kernel size of 9. The maximum MCC increased accordingly across kernel sizes as well from 0.264 at 3 to 0.283 at 9. Across all configurations, MCC scores spanned from 0.094 to 0.283

Furthermore, model size increased almost proportionally with kernel size, as shown in Figure C.11, from mean model sizes of 12.8 kB at a kernel size of 3 to 36.5 kB at 9. The pairwise win-rate matrix in Figure C.10a shows that higher kernel sizes achieved higher win rates in most pairwise comparisons, with the `kernel_size` of 9 outperforming sizes 3, 5, and 7 in 83.6%, 72.5%, and 58.8% of comparisons, respectively. Figure C.12 illustrates the relationship between model size and MCC across the different `kernel_size` values.

Dilation Figure 3.12 presents the distribution of MCC scores across different `dilation` values. The mean and median MCC values were highest for `dilation` values of 2 and 4, reaching 0.249/0.251 and 0.250/0.250, respectively. Across all configurations, MCC scores ranged from 0.094 to 0.283, with the maximum MCC increasing from 0.275 at `dilation` 1 to 0.283 at `dilation` 2.

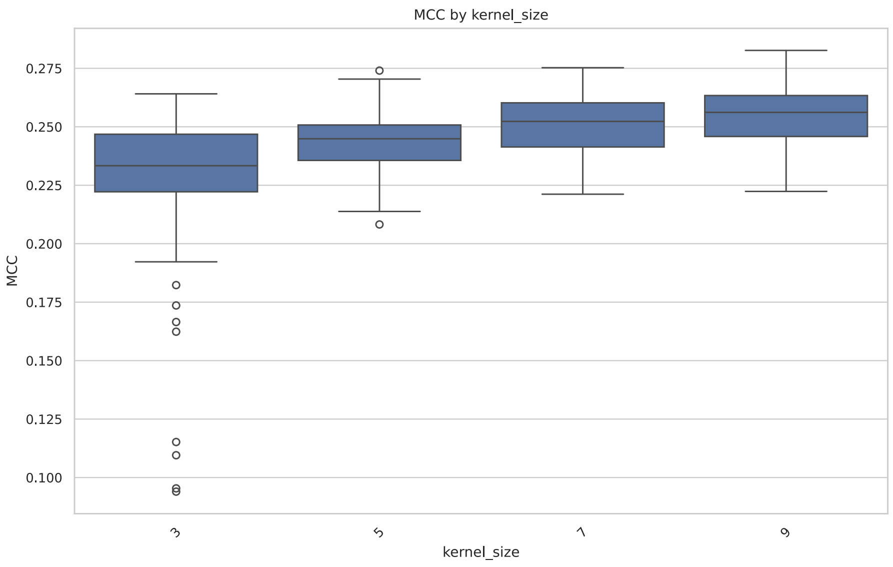


Figure 3.11: Box plot showing the distribution of MCC scores for different kernel sizes across 256 configurations

Model sizes were identical across all `dilation` values. The pairwise win-rate matrix in Figure C.10b shows that models with `dilation` values of 2 and 4 achieved higher win rates in most pairwise comparisons, with `dilation` 2 outperforming `dilation` 1 and 8 in 64.4% and 65.6% of comparisons, respectively. Figure C.13 illustrates the relationship between model size and MCC across the different `dilation` configurations.

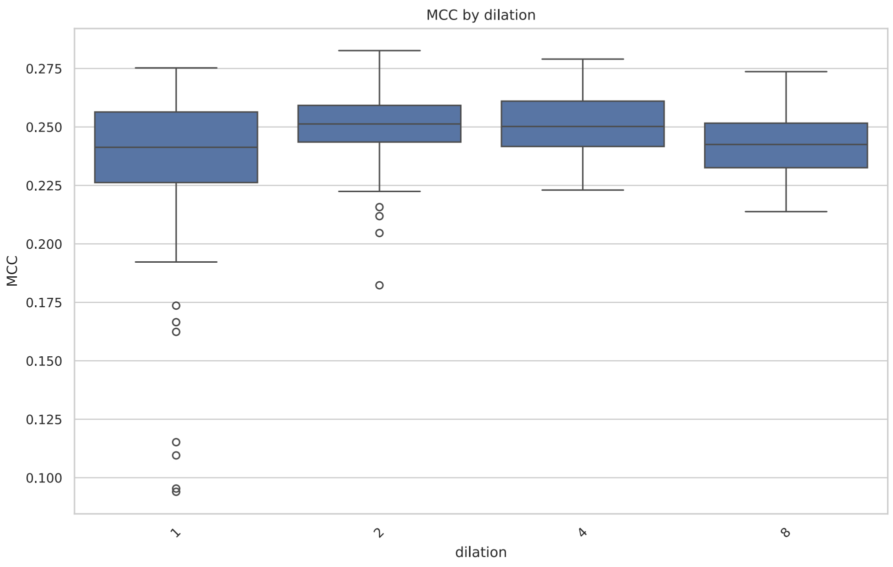


Figure 3.12: Box plot showing the distribution of MCC scores for different dilation factors across 256 configurations

3.3.1.5 Activation Layer

Figure 3.13 presents the distribution of the change in MCC between models trained with a `ReLU` and a `SiLU` activation layer. Out of the 192 analysed configurations, 109 configurations (56.8%) achieved higher MCC with the `ReLU` activation, while 83 configurations (43.2%) achieved higher MCC with the `SiLU` activation. The difference in MCC between the two activations (`SiLU` − `ReLU`) ranged from −0.0266 to 0.0259, with a mean/median difference of −0.0018/−0.0015.

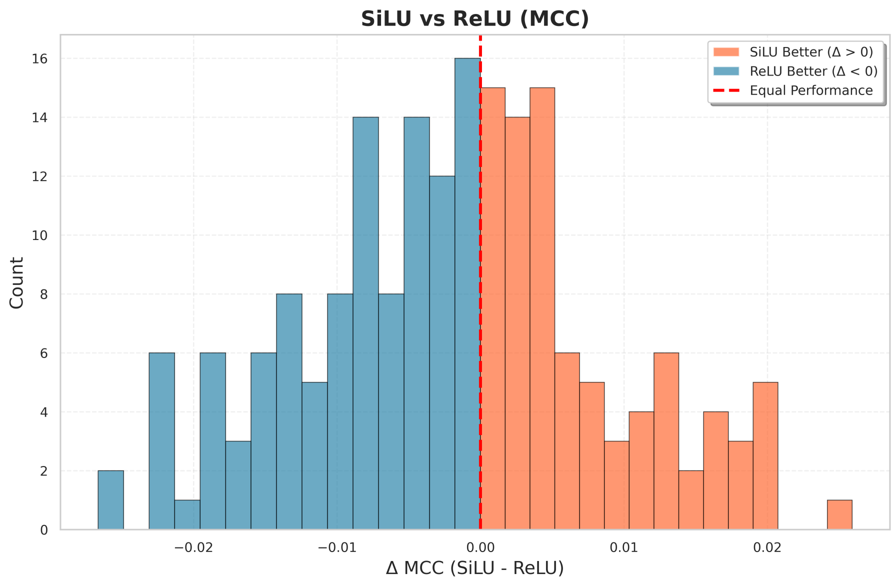


Figure 3.13: Distribution of the change in MCC between models trained with a `ReLU` and `SiLU` activation layer. A total of 109 configurations (56.8%) achieved higher MCC with the `ReLU` layer, while 83 configurations (43.2%) performed better with the `SiLU` layer.

3.3.2 Dilated CNN

Second, the parameters of the dilated CNN model family are discussed, as outlined in subsubsection 2.6.3.2.

In this analysis, the following configuration parameters of the dilated encoder–decoder CNN are evaluated individually: the number of dilated layers, bottleneck channel width, bottleneck kernel size and base dilation factor, encoder channel widths and kernel sizes, and the degree of encoder–decoder parameter similarity (subsubsection 2.6.3.2). The subsequent subsections present the corresponding results for each parameter, with performance reported in terms of MCC distributions, model sizes, and pairwise win-rate comparisons. An overview of the numerical results is provided in Table D.3.

3.3.2.1 Bottleneck stage

This part presents the results for the bottleneck stage of the Dilated CNN. The influence of the number of dilated layers and the bottleneck channel width is outlined in this subsection.

Number of dilated layers Figure 3.14 shows the influence of the number of dilated layers on model performance within the dilated model family. The mean/median MCC values increased from 0.294/0.292 for three dilated layers to 0.315/0.307 for five layers, after which a slight decline, 0.308/0.307, was observed for seven layers.

The relationship between model size and MCC, shown in Figure C.14, indicates a consistent increase in model size with a higher number of dilated layers, with mean model size rising from 209 kB for three layers to 380 kB for five layers and 452 kB for seven layers. Since configurations exceeding 2 MB were excluded, the model size distribution for seven layers is truncated, as illustrated in Figure C.17a, and therefore cannot be directly compared to the smaller architectures.

Pairwise comparisons in the win-rate matrix (Figure C.16a) confirm that models with five dilated layers outperform both smaller and deeper configurations. Win rates of 76.9% were achieved against three-layer models and 57.4% against seven-layer models, while seven-layer models outperformed three-layer models in 71.3% of comparisons.

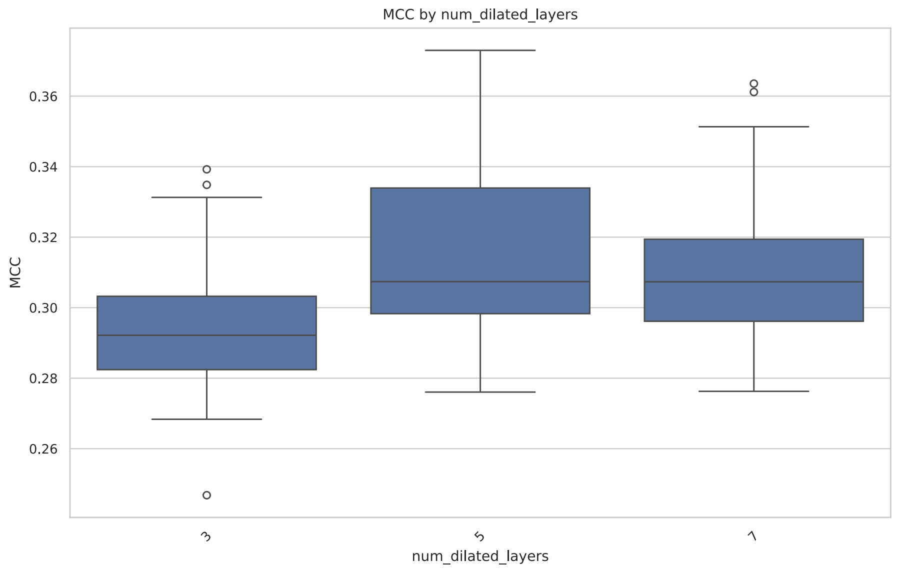


Figure 3.14: Box plot showing the distribution of MCC scores for different dilation factors across 282 configurations

Channel amount Figure 3.15 shows the results for varying bottleneck channel widths within the dilated CNN family. The mean/median MCC values consistently increased with larger bottleneck widths, ranging from 0.288/0.288 at 16 channels to 0.327/0.328 at 128 channels. Correspondingly, the model size increased from a mean of 31 kB at 16 channels to 1017 kB at 128 channels, as shown in Figure C.15 and Figure C.17b.

Pairwise win-rate comparisons in Figure C.16b indicate that models with higher bottleneck widths consistently achieve higher MCC values across comparisons. Models with 128 bottleneck channels outperformed all smaller configurations, achieving win rates of 93.9%, 85.5%, and 72.5% against models with 16, 32, and 64 bottleneck channels, respectively.

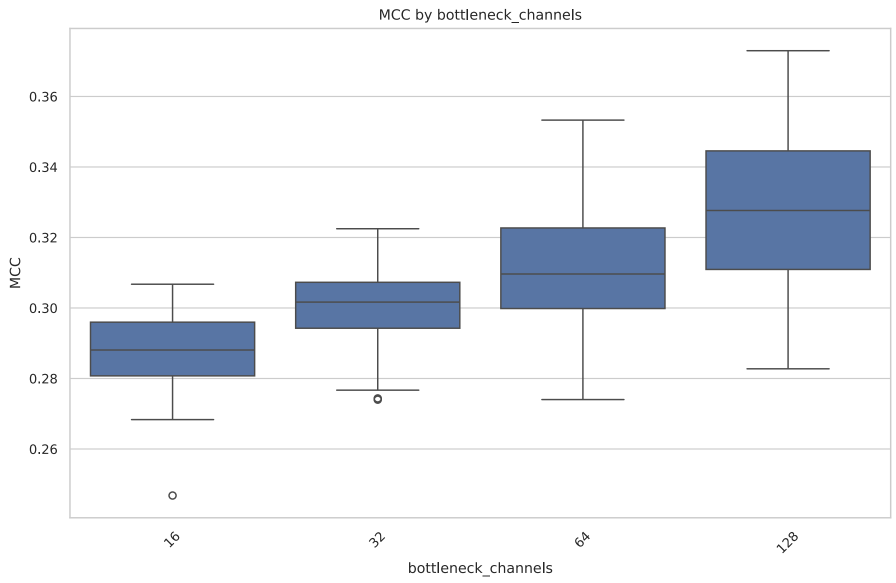


Figure 3.15: Box plot showing the distribution of MCC scores for different `bottleneck_channel` values across 282 configurations.

3.3.2.2 Convolution block

This part presents the results for the convolution block parameters within the dilated CNN architecture. The influence of kernel size and base dilation is shown.

Kernel Size Kernel size results are in accordance with the observations from the classic CNN family, showing no notable deviations in performance trends. For completeness, the corresponding figures are included in Figure C.18 to Figure C.20. As these results are largely consistent with previous findings, no further analysis is provided.

Base Dilation Figure 3.16 presents the distribution of MCC scores across different `base_dilation_dc` values. The mean/median MCC values were highest for `base_dilation_dc` = 2 (0.301/0.300), while the remaining configurations achieved slightly lower results, ranging from 0.290/0.288 at 1 to 0.296/0.296 at 3 and 0.294/0.296 at 4. Across all configurations, MCC values spanned approximately from 0.27 to 0.32.

Model sizes remained identical across all dilation settings, with mean/median of 75.8 kB/57.0 kB and a total range of 18.0–177.0 kB. The win-rate matrix in Figure C.19b furthermore shows that `base_dilation_dc` = 2 outperformed the other configurations in the majority of pairwise comparisons, 63–78% win rate. Finally, Figure C.22 illustrates the relationship between MCC and model size across different `base_dilation_dc` values.

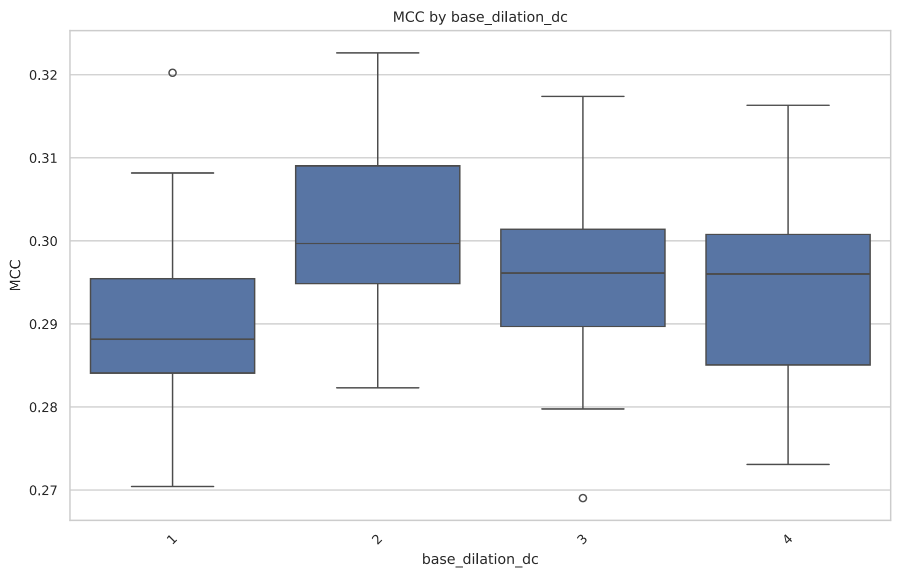


Figure 3.16: Box plot showing the distribution of MCC scores for different `base_dilation_dc` values across 128 configurations.

3.3.2.3 Encoder architecture

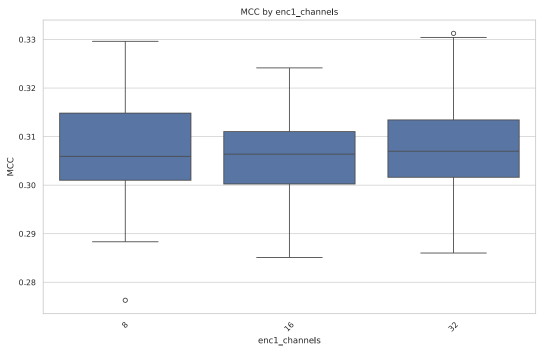
This part presents the results for the encoder architecture of the dilated CNN model family. Two components are evaluated in the two convoltuional layers in the encoder stage: the number of channels and the kernel window.

Number of channels Figure 3.17 summarises the performance results for varying encoder channel sizes in the first and second encoder convolutional layers.

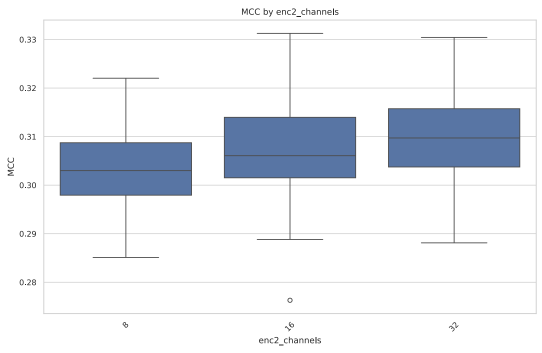
For the first encoder stage (`enc1_channels`), the mean/median MCC values remained constant across all settings at 0.307/0.306, with minimum and maximum values ranging from 0.276 to 0.331. Model sizes increased with the number of channels, with mean/median values of 63.8/57.3 kB, 70.0/63.3 kB, and 82.4/77.2 kB for `enc1_channels` = 8, 16, and 32, respectively.

For the second encoder stage (`enc2_channels`), a slight upward trend was observed in mean/median MCC values, increasing from 0.303/0.303 at 8 channels to 0.310/0.310 at 32 channels. Across all configurations, MCC values spanned approximately from 0.276 to 0.331. Model sizes followed a similar trend with increasing channels, with mean/median values of 55.1/50.1 kB, 67.8/63.8 kB, and 93.4/91.0 kB for `enc2_channels` = 8, 16, and 32, respectively. The total range of model sizes was 19.4–172.4 kB.

Pairwise comparisons in Figure C.23a and Figure C.23b show the relative win rates between configurations. For `enc1_channels`, win rates remained close to equal across all comparisons (47–53%), while for `enc2_channels`, higher channel counts achieved increasing win rates (up to 71%) against lower settings. Figure C.24 and Figure C.25 further visualise the relationships between MCC and model size for different encoder channel amounts.



(a) Distribution of MCC scores for different `enc1_channels` values.



(b) Distribution of MCC scores for different `enc2_channels` values.

Figure 3.17: Box plots showing the distribution of MCC scores across dilated CNN models for varying encoder channel sizes.

Encoder kernel window Figure 3.18 displays the performance results for varying encoder kernel sizes in the first and second encoder convolutional layer.

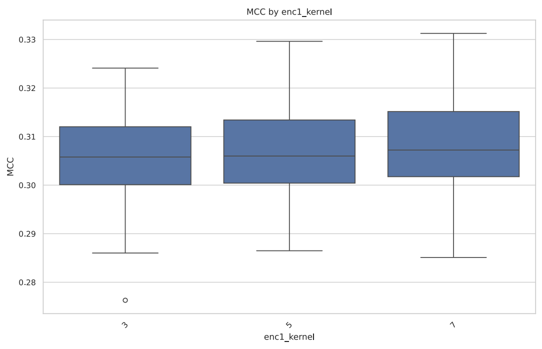
For the first encoder stage (`enc1_kernel`), the mean/median MCC values remained constant across kernel sizes, with values of 0.306/0.306 at 3, 0.306/0.306 at 5, and 0.308/0.307 at 7. Minimum and maximum MCC values ranged from 0.276 to 0.331, with a tiny increase observed in the maximum values from 0.324 at kernel size 3 to 0.331 at kernel size 7. Model sizes were highly similar across settings, with mean/median values of 71.9/65.5 kB, 72.1/65.6 kB, and 72.2/65.8 kB for kernel sizes 3, 5, and 7, respectively.

For the second encoder stage (`enc2_kernel`), the mean/median MCC values showed minimal variation across kernel sizes, with 0.306/0.306 at 3, 0.307/0.306 at 5, and 0.307/0.307 at 7. The minimum and maximum MCC values ranged from 0.276 to 0.331, with the maximum values remaining rather constant across kernel sizes. Model sizes increased slightly with kernel size, with mean/median values of 69.4/62.9 kB, 72.1/66.6 kB, and 74.8/68.9 kB for kernel sizes 3, 5, and 7, respectively. Model size values ranged from 19.4 kB to 172.4 kB.

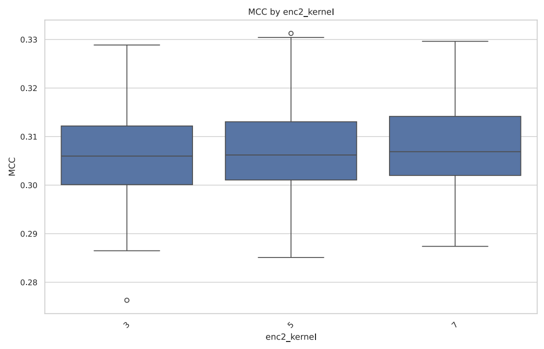
Pairwise comparisons in Figure C.27a and Figure C.27b show balanced win rates across different configurations. For the first encoder stage, win rates ranged from 44–56% between kernel sizes, while in the second encoder stage they ranged from 45–55%. Figure C.28 and Figure C.29 illustrate the relationships between MCC and model size for different encoder kernel sizes, and Figure C.30a–Figure C.30b show the corresponding model size distributions.

3.3.2.4 Decoder

Table 3.2 summarises the results for the different decoder configurations, comparing models in which the decoder was constructed symmetrically to the encoder against models using independently selected decoder parameters. Across the 256 evaluated configurations, using decoder parameters that matched the encoder resulted in slightly higher MCC values than non-similar settings. Kernel similarity increased the mean/median MCC from 0.3066/0.3066 to 0.3079/0.3080, channel similarity



(a) Distribution of MCC scores for different `enc1_kernel` values.



(b) Distribution of MCC scores for different `enc2_kernel` values.

Figure 3.18: Box plots showing the distribution of MCC scores across dilated CNN models for varying encoder kernel sizes.

from 0.3065/0.3068 to 0.3082/0.3073, and full (kernel & channel) similarity from 0.3067/0.3068 to 0.3103/0.3077.

Table 3.2: Summary of MCC results for encoder-decoder similarity comparisons across all 256 configurations. Similar configurations achieve slightly higher mean and median MCC than non-similar ones.

	MCC			
Similarity Type	**Mean**	**Median**	**Min**	**Max**
Kernel Similar	0.3079	0.3080	0.2911	0.3284
Kernel Not Similar	0.3066	0.3066	0.2806	0.3282
Channel Similar	0.3082	0.3073	0.2883	0.3231
Channel Not Similar	0.3065	0.3068	0.2806	0.3284
Both Similar	0.3103	0.3077	0.3001	0.3230
Not Both Similar	0.3067	0.3068	0.2806	0.3284

3.3.3 Pre- and post-processing

In this subsection, the effects of peak refinement and non-maximum suppression (NMS) are presented.

3.3.3.1 Peak Refinement

Figure 3.19 illustrates the performance difference in MCC between models trained and evaluated with and without peak refinement. Each bar represents a matched configuration pair, showing the change in MCC ($\Delta\text{MCC} = \text{MCC}_{\text{adjustment}} - \text{MCC}_{\text{original}}$).

Across all analysed configurations, peak adjustment consistently improved model performance. The mean/median MCC increased from 0.269/0.262 without adjustment to 0.417/0.404 with adjustment. The corresponding standard deviations were 0.0259 and 0.0645, indicating slightly increased variability among adjusted results.

The improvement was observed in all 96 pairs, with positive ΔMCC values ranging from 0.04 to 0.29.

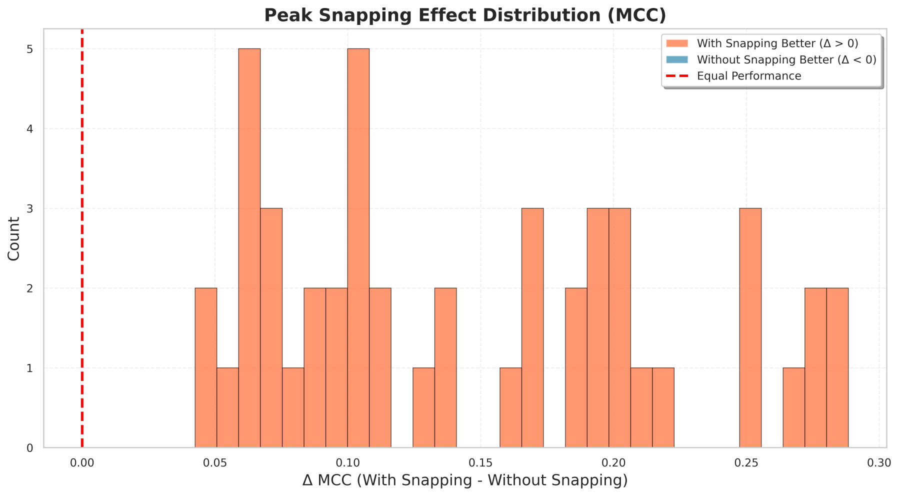


Figure 3.19: Histogram showing the MCC difference ($\Delta\text{MCC} = \text{MCC}_{\text{adjusted}} - \text{MCC}_{\text{original}}$) for all 96 configurations. Positive values indicate improved performance when applying peak adjustment.

3.3.3.2 Non-Maximum Suppression (NMS)

Figure 3.20 presents the distribution of MCC scores for configurations without NMS, and with NMS thresholds of 5 and 14.

For the case without NMS, the mean/median MCC was 0.2690/0.2617, ranging from 0.2249 to 0.3148. With NMS set to 5, the mean/median MCC decreased to 0.1604/0.1573, spanning 0.1072–0.2066. Similarly, for NMS set to 14, the mean/median was 0.1611/0.1564, ranging between 0.1073 and 0.2077. The corresponding standard deviations were 0.0259, 0.0308, and 0.0311, respectively.

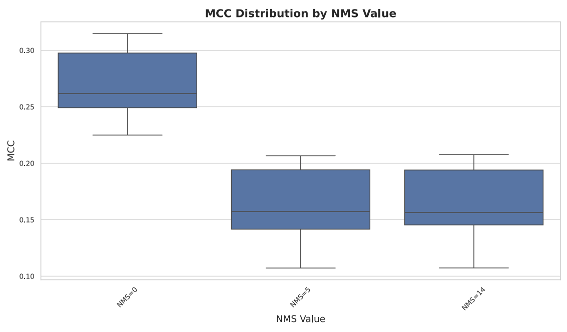


Figure 3.20: Distribution of MCC scores for models with NMS disabled, and with NMS thresholds of 5 and 14.

3.4 Embedded Validation

To ensure the correctness of the converted models, validation was performed both on a PC and directly on the embedded target. The evaluation compares the numerical outputs of the original model, the generated embedded model executed on the host PC, and the embedded model running on the embedded device. The results are summarised in Table 3.3. For the embedded system, inference was executed at a clock frequency of 480 MHz. The measured CPU cycle counts are stated in Table 3.4.

Table 3.3: Cross-validation results comparing the original model and the C-model on both the host PC and the embedded target.

	PC			**Embedded**		
	($\times 10^{-6}$)			($\times 10^{-6}$)		
Configuration	**MAE**	**RMSE**	**CS**	**MAE**	**RMSE**	**CS**
Standard CNN	0.678	0.888	1.000000	0.980	1.232	1.000000
Dilated CNN	1.018	1.384	1.000000	0.877	1.171	1.000000

Table 3.4: Embedded inference performance for both model families, reporting individual CPU cycle measurements and their corresponding execution times at a 480 MHz clock, followed by the averaged values.

Model	**Cycles**	**Time [ms]**
Standard CNN		
Run 1	45,161,538	94.087
Run 2	45,149,343	94.061
Run 3	45,154,339	94.072
Average	**45,155,073**	**94.073**
Dilated CNN		
Run 1	95,203,877	198.341
Run 2	95,222,106	198.379
Run 3	95,194,127	198.321
Average	**95,206,703**	**198.347**

These error rates resulted in no differences between the identified peaks on the original model and the embedded model executed on an embedded device.

Discussion 4

4.1 PPG Dataset

The collected PPG dataset forms the foundation of this research. It includes measurements from multiple anatomical ear locations and activity conditions, resulting in a diverse dataset suitable for detailed analysis. This subsection focuses on describing and visually interpreting the dataset characteristics, highlighting differences in signal quality across locations and conditions. No machine learning results are included here, these will be discussed in the the comparative section (section 4.2). Instead, the discussion in this section is limited to qualitative waveform observations and statistics (skewness) to provide an initial assessment of signal behaviour.

First of all, one of the most obvious observations from Figure 3.1 and Figure 3.2 is the clear difference in signal quality between different in-ear sensor locations. For instance, the deep meatus (Figure 3.1a) and concha (Figure 3.1b) show relatively clean PPG waveforms, where most distortions are of a low-frequency nature, visible as gradual baseline drifts. In contrast, the tragus and shallow meatus locations (Figure 3.1c and Figure 3.1d) show a combination of both low- and high-frequency distortions, resulting in slow baseline shifts, combined with jagged, irregular fluctuations in the signal.

Moreover, noticeable differences in peak-to-peak amplitudes are observed between locations, indicating variations in the signal-to-noise ratio (SNR). These effects become even more pronounced under dynamic (walking) conditions. While the deep meatus and concha signals still retain a recognisable PPG-like waveform, the signals from the tragus and shallow meatus degrade substantially, making the underlying cardiac waveform no longer identifiable.

The skewness summary in Table 3.1 only partially supports the visual observations. [29] suggested a fixed threshold at 0 to seperate ”excellent” waveforms from ”unacceptable” PPG waveforms, with positive skewness indicating good-quality signals and negative skewness indicating poor quality. However, our results show that this criterion is not enough in practice. Several locations exhibit skewness values close to zero or even clearly positive while still producing visually degraded or unstable PPG waveforms. This illustrates that this binary threshold alone does not reliably determine overall signal quality.

For the deep meatus locations, median/mean skewness values are positive and consistent in both seated (0.092/0.111) and moving (0.099/0.123) conditions. Interestingly enough, the moving condition results in more right-skew than the seated condition, although IQR and SD is higher. While this is not expected, these changes could be attributed to a different fitting of the sensor module inside the ear and should not directly lead to the conclusion that the moving condition is preferred to the seated condition. These positive and consistent skewness values are furthermore expected by our visual interpretation of this sensor location.

Continuing to the shallow external auditory meatus location, the mean/median skewness is negative in both seated ($-0.030/-0.065$) and moving ($-0.117/-0.140$) condition, with the moving condition significantly more negative than the seated condition, which aligns with our visual observation.

The concha location is quite inconsistent between non-moving and moving conditions. Where the seated condition results in the most right-skewed median/mean values (0.260/0.201) with the lowest variability in the dataset (IQR/SD of 0.334/0.311), the moving condition results in a slight left-skewed signal ($-0.024/-0.047$), where the IQR value becomes the highest in the dataset (0.924), with the standard deviation staying a bit more stable compared to the other sensor locations (0.598). It can be noted that motion-induced distortions have a significant effect on this location, both from visual interpretation as well as the skewness factor.

At last, the tragus location results in inconsistencies between our visual interpretations and the results from the skewness factor. Both non-moving and moving signals are right-skewed, with median/mean values of 0.085/0.117 and 0.065/0.073, respectively. In essence, taking the > 0 skewness threshold should imply this is a good signal. Although this appears true for the non-moving condition, where a recognisable PPG waveform can still be observed, the moving condition does not show a clear PPG-like structure despite maintaining a positive skew. Moreover, the variability values are notably high, with an IQR of 0.800 for both conditions and standard deviations for non-moving and moving of 0.780 and 0.620, respectively. These relatively large spreads indicate substantial variability across analysis windows, suggesting that the waveform morphology is unstable and inconsistent. Therefore, while the positive skewness values might initially suggest acceptable signal quality, the accompanying large variance contradicts this interpretation. This highlights that skewness alone cannot reliably characterise signal quality at the tragus, particularly under motion, and that additional metrics are needed for a more accurate assessment.

All in all, taking the visual observations and skewness factors into account, the following hierarchy of sensor placement can be derived:

1. **Deep Meatus:** Consistent PPG waveform at rest and during movement
2. **Concha:** Excellent waveform at rest, discernible PPG waveform during movement, negative skew during movement
3. **Tragus:** Decent PPG waveform at rest, major distortions during movement
4. **Shallow Meatus:** Low- and high-frequency noise at rest, minor discernible PPG waveform at rest, unrecognisable waveform during movement, with significant negative skewness

4.2 Comparative analysis

The comparative results in Figure 3.4 and Figure 3.5 confirm the impact of sensor placement on model accuracy. A clear performance increase is observed from the tragus location to the deep external auditory meatus. These results mirror the earlier signal quality findings shown in section 3.1 and discussed in section 4.1. The deep external auditory meatus achieved the highest mean/median MCC (0.3041/0.3078), followed by the concha (0.2567/0.2598), shallow external auditory meatus (0.1612/0.1690), and tragus (0.0646/0.0621).

The pairwise win rate matrix further supports this ranking, with the deep external auditory meatus outperforming all other locations (92–100% win rates), and the concha consistently surpassing the shallow external auditory meatus and tragus locations.

From these results, assumptions could be drawn that the deeper regions of the external auditory meatus provide greater mechanical stability, leading to cleaner PPG measurements, more consistent feature extraction, and consequently higher model performance. Furthermore, sub-optimal signal acquisition at the tragus and shallow external auditory meatus locations result in poorer outcomes, which could be the result of a lesser secure fit. This emphasises that fundamental factors such as a secure sensor fit still remain important for reliable model inference. Although the deep external auditory meatus location provides the best performance, the concha could be used a an practical alternative location when a custom-fit deep insertion is not feasible, though with some loss in accuracy.

Overall, model outcomes strongly correlate with the qualitative and statistical dataset analyses. The resulting performance ranking is summarised as follows:

1. External auditory meatus (Deep)
2. Concha
3. External auditory meatus (Shallow)
4. Tragus

4.3 Standard CNN configuration parameters

Section 3.3 introduces different results for the range of configuration parameters available for the machine learning models. In principle, exploration of all parameter combinations would provide the most comprehensive understanding of model behaviour. However, such an approach is computationally infeasible due to the exponential growth of the search space. Consequently, this section focuses on analysing the results from a controlled subset of configurations to identify parameter settings that result in promising performance. The insights obtained can be used to limit the subsequent search space, thereby improving the efficiency of computational resource utilisation in later experiments.

4.3.1 Batch Normalisation

The first configuration parameter to be analysed is the batch normalisation. This parameter proved to have a direct, significant impact on the performance of the resulting ML model. Out of the 32 tested models, only two models achieved a marginally higher MCC without batch normalisation, while all other models benefited from minor to substantial improvements with the application of batch normalisation. From these results, it can be assumed that batch normalisation enhances training stability and reduces internal covariate shift, which proves to be beneficial in almost all configurations. The few configurations that did not improve with batch normalisation may be attributed to stochastic factors such as suboptimal weight initialisation or insufficient training epochs rather than an actual disadvantage obtained from batch normalisation. Overall, the inclusion of batch normalisation is strongly recommended in the model architecture, given its consistent positive impact on predictive performance across nearly all tested configurations.

4.3.2 Pooling strategies

Following subsections will discuss the configuration parameters that are related to different pooling strategies, namely the pooling frequency, determined by a `pool_every` value, and the pooling type. The corresponding results have been outlined in subsubsection 3.3.1.2.

4.3.2.1 Pooling frequency

From the results, it can be observed that increasing the pooling interval from one to two convolutional blocks resulted in a clear performance improvement. A likely explanation is that pooling after every block reduces the temporal resolution too quickly/aggressively, causing the model to lose relevant details that are important for identifying PPG peaks. Using a slightly larger interval (`pool_every` = 2) could allow the network to first extract a more complete representation before reducing the feature map length, leading to better overall performance.

Further increases of the pooling interval to four or eight blocks did not result in additional improvements, but results in a higher variability. A number of models with `pool_every` values of four and eight result in substantially lower MCC scores compared to the mean, resulting in a larger spread and low-performing outliers. This could however be the result of the corresponding architectures not containing any pooling layers, as the pooling frequency could have exceeded the number of convolutional blocks in the architecture. This does however lead to the same conclusion that pooling frequency is still an important factor that should not be overlooked. A `pool_every` value of two demonstrates a minor increase in mean/-median than the values of four and eight and fewer low outliers, suggesting more consistent convergence across configurations. It should also be noted that lower pooling frequencies (so higher `pool_every` values) leads to higher computational cost, as the temporal length of the intermediate feature maps is not decreased, resulting in significantly more computational operations performed during convolution. Therefore, it is recommended to use the highest pooling frequency that does not compromise performance, as this provides the most computationally efficient configuration, an important factor in embedded environments. In this case, a `pool_every` value of 2 offers the best trade-off between performance and efficiency.

4.3.2.2 Pooling type

For the different pooling layer types, average, L_p-norm, and max pooling, only minor variations in performance were observed. All methods achieved comparable mean/median MCC values, as shown in Table D.2. While slight differences can be seen in the minimum and maximum MCC values across pooling types, these deviations are relatively small and likely reflect normal variability between configurations rather than a systematic performance effect. This interpretation is further supported by the win rate matrix in Figure C.4b, where none of the pooling types consistently outperforms the others (all win rates remain close to 50%). Overall, the results indicate that the choice of pooling type has no substantial influence on model performance in this context.

Although performance differences between pooling types are negligible, their computational costs differ. Since the window length of the pooling layer is set to 2, each pooling operation cost is as follows:

- **Max pooling:** 1 comparison (Equation 2.22).
- **Average pooling:** 1 addition and 1 bit shift (Equation 2.23).
- L_p**-norm pooling:** 2 elementwise power operations 1 addition, 1 multiplication/division (depending on the implementation) and 1 p-th root calculation; for p=2, with compiler optimisation, this could reduces to 2 multiplications 1 addition, 1 bit shift and 1 square root. (Equation 2.24).

Instruction-level costs are processor-dependent, but for embedded-class microcontrollers such as the ARM Cortex-M0, both addition (`ADD`) and comparison (`CMP`) operations are executed in a single clock cycle [60], with multiplication, division and square root operations usually taking up much more cycles. While different architectures have changes in implementation, one can assume that most competitive microcontrollers on the market all have single clock cycle `ADD` and `CMP` instructions, with the other relevant arithmetic operations taking up more time. Given that all pooling types achieved comparable MCC performance, but differ substantially in computational cost, max pooling would be the most suitable choices when computational efficiency or energy consumption is a priority.

4.3.3 Channels

Channel configuration defines the model's representational capacity and size. This section discusses how the number of initial channels (`base_channels`) and their growth rate (`double_every`) affect performance, while identifying balanced settings for accuracy and efficiency in embedded use.

4.3.3.1 Base Channels

Increasing `base_channels` result ins small but consistent MCC gains. The mean/median MCC increases from 0.230/0.236 at 8 channels to 0.243/0.251 at 64 channels (Table D.2, Figure 3.9). The best single model (`MCC` 0.292) appears in the widest group. These gains, however, require a much larger footprint, mean model size increases from 27.4 kB (8 channels) to 380.5 kB (64 channels), roughly a 14× increase. The relatively small increase in MCC shows that increasing the `base_channel` can certainly be a viable option to increase performance, but must be balanced with its size increase, especially in an embedded environment where memory is scarce.

4.3.3.2 Channel growth frequency

More frequent growth (smaller `double_every`) shows a small advantage in pairwise comparisons, while average performance remains similar across settings. Specifically, the 1 configuration beats 8 in 55.0% of pairs and 2 beats 8 in 55.9% (Figure C.6b). Furthermore, mean/median MCC values remain close, 0.236/0.244 to 0.239/0.249 across `double_every`=1,2,4,8 (Table D.2, Figure 3.10). The best MCC, 0.292, is reached by a 1.2 MB model within the `double_every`=2 set, while for `double_every`=1 the best is 0.284 at 200 kB (Figure C.9).

At the same time, model size decreases systematically with less frequent scaling (Figure C.7b), creating a clear trade-off between minor performance improvements and model size. The results suggest that overly aggressive scaling shows decreasing returns relative to the corresponding increase in model size. Consequently, `double_every`=2 offers a practical balance between maintaining strong performance while avoiding unnecessary growth in parameter count. In more constrained environments, higher `double_every` values may be preferred to minimise memory usage, at the expense of a small reduction in accuracy.

4.3.4 Convolution layer

The convolution layer determines how temporal features are captured across the PPG signal. Key parameters such as `kernel_size` and `dilation` are discussed in this subsection.

4.3.4.1 Kernel Size

Larger `kernel_size` values result in small but consistent performance gains. The mean/median MCC increased from 0.224/0.233 at size 3 to 0.255/0.256 at size 9, with the latter outperforming smaller kernels in up to 83.6% of pairwise comparisons (Figure C.10a). This indicates that broader receptive fields certainly improve temporal feature extraction and is therefore one of the key components in improving the model performance. Moreover, model sizes rose proportionally from 12.8 kB to 36.5 kB (Table D.2), implying a clear trade-off between accuracy and model size. It must further be noted that a `kernel_size` value of 3 exhibits many outliers beyond the 1.5$*$ IQR range in the MCC distribution (Figure 3.11), indicating inconsistent performance across configurations. In contrast, larger kernel sizes show more compact distributions, suggesting greater stability in model behaviour. All in all, kernel sizes of 5–7 offer a practical balance for most embedded deployments with respect to performance, stability and model size, where higher values could be used in environments with more memory capacity than commonly found in embedded devices.

4.3.4.2 Dilation

Performance peaked for moderate dilation factors. Mean/median MCC values reached 0.249/0.251 and 0.250/0.250 for `dilation`=2 and 4, outperforming both smaller and larger values (Figure C.10b). This suggests that moderate dilation effectively extends temporal context without excessively spacing sampling and in turn losing important characteristics. As model size and computational complexity stay constant with different dilation values, moderate dilation factors are recommended with values of 2 and 4.

4.3.5 Activation Layer

Figure 3.13 shows that the performance differences between `ReLU` and `SiLU` in the activation layer are relatively small. Across the 192 paired configurations, differing only in activation, `ReLU` resulted in a higher MCC in 109 cases (56.8%), while `SiLU` was better in 83 cases (43.2%). The pairwise MCC differences (`SiLU` − `ReLU`) ranged from −0.0266 to 0.0259, with a mean/median of −0.0018/−0.0015. In other words, while pairwise differences in the MCC can make a small difference, the mean/median values of the difference is incredibly close to zero.

Technically speaking, these results indicate a minor advantage for `ReLU` (both in win rate and average pairwise difference), but the magnitude of the advantage is extremely modest. Moreover, `SiLU` does attain the higher MCC in still a substantial fraction of cases (43.2%), implying that either activation can be part of a competitive configuration depending on the remaining hyperparameters.

From an embedded deployment perspective, because the two activations produce almost similar MCCs on average while `ReLU` is computationally simpler to evaluate, a reasonable strategy is to choose a `ReLU` activation layer when compute budget is a primary concern. If the goal is to maximise absolute peak performance and hyperparameter search compute budget permits, including both `ReLU` and `SiLU` in the search remains justified, as `SiLU` is occasionally the best-performing choice for specific configurations.

4.4 Dilated CNN configuration parameters

This section discusses the configuration parameters specific to the dilated CNN model family, including the bottleneck depth and width, base dilation factor, encoder settings, and decoder structure.

4.4.1 Bottleneck Stage Parameters

For the bottleneck stage, the number of layers and their corresponding channel width will be discussed.

4.4.1.1 Number of layers

A clear performance optimum is observed at `num_dilated_layers`=5, achieving a mean/median MCC of 0.315/0.307 and a maximum of 0.373 (Figure 3.14). As model size increases with network depth, configurations with seven layers offer no benefit, being larger yet slightly less accurate. This behaviour is fully expected given the 4-second window, 100 Hz sampling frequency, and the encoder downsampling factor of 4, which together reduce the temporal input length of the bottleneck stage to 100 samples. The dilation grows exponentially, making the effective kernel window quickly exceed this available temporal context. At five layers, the dilation has already expanded to $2^5 = 32$, and with a kernel size of 3, the receptive window becomes $3 \times 32 = 96$ samples, already covering nearly the entire input length. Adding further layers increases the receptive field of the kernel far beyond the 100-sample input, making additional dilated convolutions ineffective. In memory-constrained environments, three layers remain a viable lightweight alternative, but five layers provide the best trade-off between performance and model size, indicating that capturing the full temporal context does lead to improved performance. This conclusion is further supported by the win rate matrix (Figure C.16a), where the five-layer models outperform both smaller and deeper configurations.

4.4.1.2 Amount of channels

Figure 3.15 perfectly shows the trade-off for the amount of channels (`bottleneck_channels`) in the model. Higher channel widths improve performance with the mean/median MCC rising from 0.288/0.288 at 16 channels to 0.327/0.328 at 128 channels. The pairwise win rate matrix in Figure C.16b confirms this trend, showing consistently higher win rates with increasing channel counts. This performance increase does however come at the cost of substantially larger models, increasing from 31 kB to over 1 MB on average with maximum model sizes for a channel width of 128 reaching past the 2 MB cutoff. Overall, this parameter has a clear and direct impact on model performance, with wider bottlenecks providing measurable gains that must be balanced against the corresponding growth in model size, especially under embedded memory constraints.

4.4.2 Convolution layer parameters

As the influence of `kernel_size` in the Dilated CNN family aligns with the findings discussed for the standard CNN models in subsubsection 4.3.4.1, only the effect of the base dilation factor is examined here.

4.4.2.1 Base dilation factor

As for the base dilation factor (`base_dilation_dc`), a value of 2 demonstrates a clear performance advantage over the other analysed settings. This is observable from the MCC distributions shown in Figure 3.16, where models with a base dilation of 2 achieve consistently higher median values. The win rate matrix in Figure C.19b further supports this trend, showing that `base_dilation_dc`=2 outperforms the other configurations in the majority of pairwise comparisons, with win rates ranging from 63.3% (versus 3) to 78.1% (versus 1). Importantly, modifying the base dilation factor does not change model size or computational complexity, making a setting of 2 the clear choice for optimising model performance.

4.4.3 Encoder architecture

The evaluation of the encoder architecture reveals that variations in both channel width and kernel size have only a minor influence on model performance, indicating that the dilated models predictive capability is largely impacted by later architectural components instead of by the complexity of the initial downsampling stage.

For the first encoder stage (`enc1_channels`), performance remains stable across all tested configurations, as illustrated in Figure 3.17 and summarised in Table D.3. The almost identical MCC distributions and balanced win rates (47%–53%) in Figure C.23a suggest that smaller channel counts already provide sufficient capacity for initial feature extraction. Increasing `enc1_channels` beyond this minimum value therefore mainly increases model size without measurable accuracy gains. Therefore, a channel amount of 8 is recommended for the initial layer.

In contrast, the second encoder stage (`enc2_channels`) exhibits a small but consistent trend favouring larger channel widths. As shown in Table D.3, mean/median MCC values rise slightly from 0.303/0.303 to 0.310/0.310 with channel count, while Figure C.23b confirms that higher configurations achieve clear pairwise advantages, with up to 71% win rates against smaller settings. This behaviour suggests that the second encoder layer benefits from additional feature channels. However, the improvements remain limited in their effect, and the increase in model size from roughly 55 kB at 8 channels to over 90 kB at 32 channels does have an impact in embedded contexts. A configuration of 16 channels therefore represents a balanced trade-off between performance and computational complexity.

The kernel size experiments (Figure 3.18) show that all tested configurations perform nearly identically across both convolutional layers in the encoder stage. The distributions of MCC values and pairwise win rates (Figure C.27a and Figure C.27b) are mostly similar, with win rates remaining within 44%–56% across all kernel sizes. As summarised in Table D.3, mean/median MCC values vary by less than 0.002 between kernel sizes 3, 5, and 7, while model sizes increase only marginally from approximately 69 kB at kernel size 3 to 75 kB at kernel size 7 for both stages. These small differences indicate that all kernel configurations extract comparable features, and that expanding the receptive field at the encoder level results in almost no measurable benefit. The similarity across settings suggests that temporal features are already well captured by smaller kernels.

4.4.4 Decoder

The evaluation of encoder–decoder similarity shows that matching the decoder parameters to those of the encoder provides a marginal performance benefit (Table 3.2). Across all 256 configurations, symmetric designs achieve slightly higher mean and median MCC values than independently configured decoders in all cases, with kernel similarity, channel similarity, and kernel & channel similarity.

Given these results, adopting a symmetric decoder is the most practical choice. Besides offering slightly better performing configurations, it also substantially reduces the dimensionality of the search space, simplifying optimisation and decreasing the computational effort required for the search space traversal. This makes similarity the preferred design.

4.4.5 Performance vs. standard

Although no explicit combined comparison between the two architecture families was included in the results section, the distributions reported for their different parameters shows that the dilated CNN performs better in most cases. Across comparable model sizes and parameter ranges, the dilated CNN consistently achieves higher mean and median MCC values than the standard CNN (easily comparable between Table D.2 and Table D.3). This indicates that the dilated bottleneck structure, which captures a larger temporal context, does provide improved peak detection in most configurations.

4.5 Pre and Post-Processing Techniques

This section discusses the effects of peak refinement and non-maximum suppression (NMS).

4.5.1 Peak refinement

The results in Figure 3.19 show a clear improvement in classification performance when peak refinement is applied during dataset preparation and evaluation. Across the configuration pairs, the MCC increased from a mean/median of 0.269/0.262 to 0.417/0.404, with all pairs demonstrating a positive ΔMCC. This indicates that a small, local correction of the label position around the ECG-derived timing can improve the learnability of the task by aligning labels more precisely to the local maxima of the peak-aligned PPG signal.

However, the improvement must be contextualised. Refining the peaks is a dataset-side operation and thus does not have direct effect at inference. Furthermore, choosing maximum values in specific windows around the ECG peaks may also set labels to artefactual maxima (motion, noise, or other error artefacts) rather than physiological peaks, which helps reported metrics yet risks teaching the model incorrect patterns.

Moreover, window size is an important control factor with significant effects. A narrow window limits artefact capture but may miss true peaks due to changing temporal factors. A wider window tolerates more severe misalignment but increases artefact inclusion risk. In this study, a small window (20 ms) offered a pragmatic balance, though the optimal choice is dataset- and hardware-dependent.

Altogether, while peak refinement is theoretically an efficient implementation for discarding inferior peaks around superior peaks, more research is needed to tune the window size and see its effect in practice on the performance.

4.5.2 Non-Maximum Suppression (NMS) Effects

As shown in Figure 3.20, enabling non-maximum suppression (NMS) results in a clear degradation in performance. The mean/median MCC values decrease substantially from 0.2690/0.2617 without NMS to 0.1604/0.1573 with NMS applied. This indicates that, under the evaluated conditions, the inclusion of NMS negatively affects overall model performance.

In the main grid search, all models were trained for 8 epochs due to computational constraints. To assess whether the observed degradation was influenced by the limited training duration, a single model was subsequently trained for 100 epochs and analysed in more detail. Figure 4.1 and Figure 4.2 show five representative signal windows from this model, comparing predictions with and without NMS. The NMS window spans 280 ms in total ($f_s = 100$ Hz, covering 14 samples before and after each detected peak).

At 8 epochs, several correctly identified peaks are suppressed because their predicted values are not the highest within the defined window. However, at 100 epochs, the opposite trend emerges, with false detections removed in favour of true positives. This suggests that models with higher predictive performance can potentially benefit from NMS to refine detections.

These findings imply that although the grid search results indicate superior performance without NMS, this outcome may depend on the model's predictive accuracy and the constraints imposed on the grid search due to computational feasibility. For less well-trained models, NMS tends to remove correct detections, whereas more accurate models may benefit from it by suppressing incorrectly predicted peaks. Further evaluation at extended training durations could provide additional insight, but was unfortunately not feasible within the current computational limits.

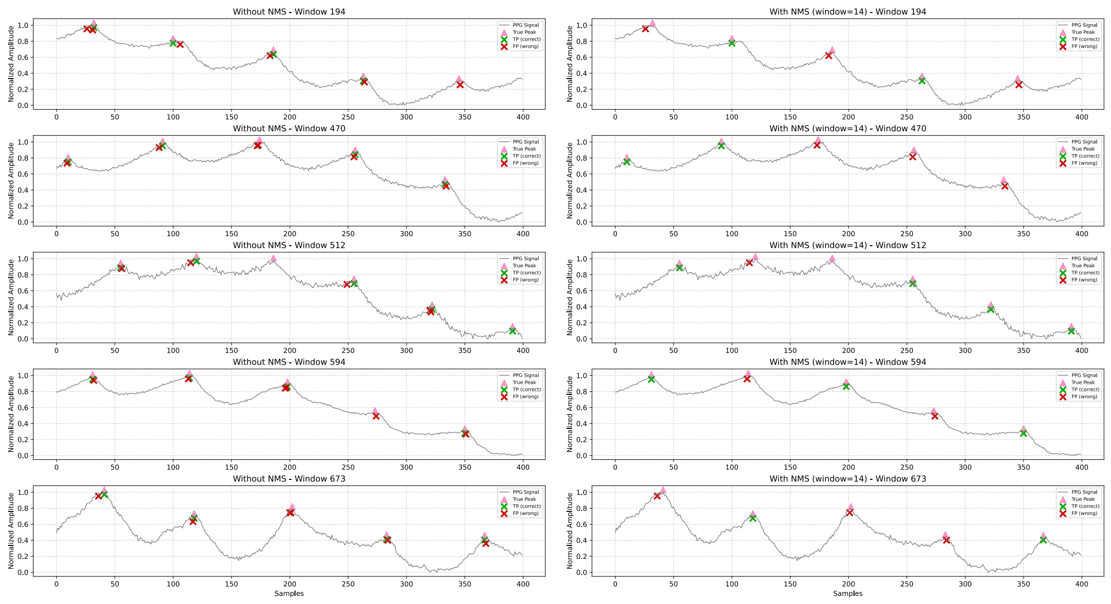


Figure 4.1: NMS in action for sample model trained with 8 epochs

4.6 Embedded validation

The embedded validation results demonstrate that the deployed models reproduce the behaviour of the original reference models with near similar results. As shown in Table 3.3, the MAE and RMSE values for both architectures remain in the order of 10^{-6}, and the cosine similarity consistently equals 1.0 across all comparisons. The

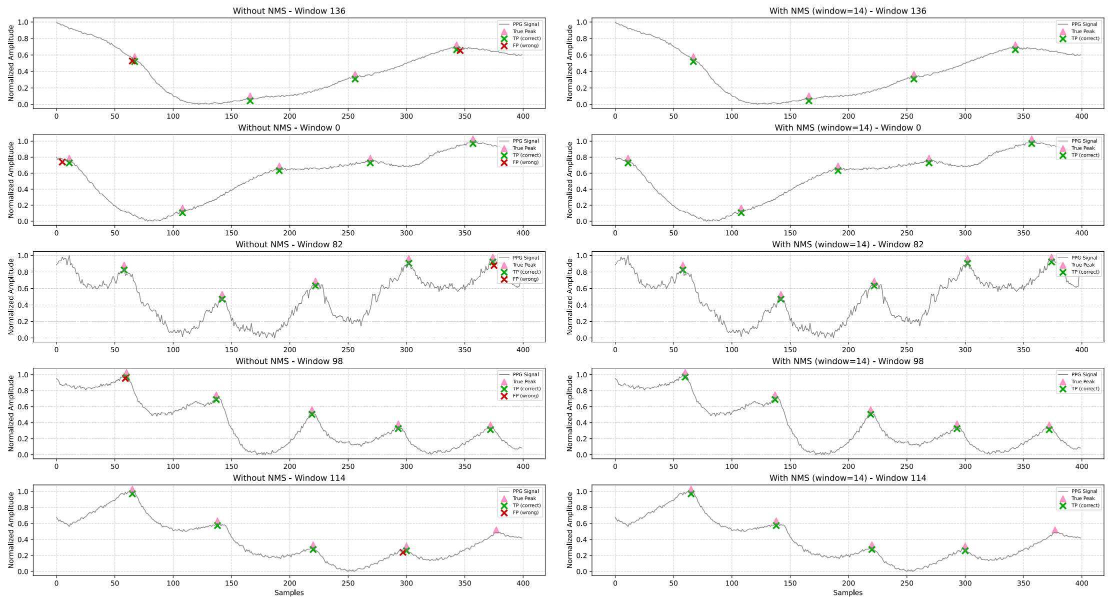


Figure 4.2: NMS in action for sample model trained with 100 epochs

remaining differences likely originate from floating-point precision changes introduced during the conversion process, non-identical machine code after compilation for different architectures (x64 vs ARM), or different hardware-specific implementations of arithmetic operations.

While the execution-time results in Table 3.4 are based on a single representative configuration per model family, meaning no general conclusions can be drawn, they do provide an indication of the achievable performance. This specific model of the standard CNN family completes inference in approximately 94,ms, while the dilated CNN instance requires around 198,ms. Given that inference operates on a 4,s window, these latencies are well within real-time requirements, and higher inference rates remain feasible for more extensive pipelines that may benefit subsequent processing stages.

4.7 Total performance

When looking at the results provided in chapter 3, one might raise the question of whether the obtained performance is adequate, as the reported MCC values remain relatively low. However, it should be noted that these results were obtained using a limited training duration of only eight epochs due to computational constraints and without the implementation of peak adjustment. When extending the training to 100 epochs and enabling peak adjustment, a specific model configuration achieved a significantly higher MCC of 0.96 compared to its previously reported 0.29. Furthermore, while the differences between individual models may appear minor in the presented results, these minor differences could amplify substantially in final performance once models are trained for a sufficient number of epochs, highlighting the importance of adequate training duration in fair model comparison.

4.8 Limitations & Future research

Although the presented research demonstrates promising results, this research, however, is subject to several limitations.

First, the limited computational resources available constrained the training duration per model. Each model was trained for only eight epochs. While this is generally sufficient to observe general learning trends, it is typically insufficient for achieving full convergence to the optimal model parameters. Consequently, the obtained results may not reflect the true performance potential of the corresponding model. Future work could include extended training in order to adequately research the performance differences of different models under full training convergence.

Second, all experiments were conducted on a single subject, a white male, due to time constraints and limited access to participants and compatible hardware. As such, the results may not generalise across individuals, given physiological, anatomical, and skin-tone differences. Future research could therefore include a more diverse participant pool to assess variability between individuals and improve generalisability.

Third, recordings were performed under controlled activity conditions to enable fair comparisons between sensor placements. While this approach ensured consistency, it also limits the applicability of the results to real-world scenarios. Variability introduced by spontaneous movements, more intense activities than currently measured (e.g., running or jumping), sweating, and environmental changes was not represented in the current dataset. Expanding the dataset to include these real-world conditions would provide valuable insights into model robustness under practical use.

Fourth, the evaluation in this thesis relied solely on sample-level metrics to ensure objectivity and reproducibility across experiments. However, this approach enforces a strict temporal boundary, where even small temporal shifts in predicted peaks are directly classified as errors. Future research could therefore explore the use of event-level metrics implemented in the framework to provide a more nuanced assessment that better captures clinically relevant performance, quantifying the potential improvements introduced by the temporal tolerance.

Fifth, small temporal alignment errors were introduced by the asynchronous measurement setup between the PPG and ECG systems. Although post-processing methods such as visual alignment and peak adjustment were applied to correct these offsets, minor inaccuracies and potential user-induced biases may remain. Future work could address this by developing fully synchronised multi-sensor acquisition systems, ensuring a temporally consistent dataset.

Finally, although the embedded validation confirmed that the converted models execute correctly on the STM32H7 microcontroller, the evaluation was limited to a single representative model per architecture. As a result, the effect of architectural choices on execution speed remains largely unexplored. Future research could therefore systematically benchmark a broader set of model configurations to characterise how parameter variations influence execution performance, and to identify opportunities for further optimisation of embedded performance.

5 Conclusion

This thesis presented the design, development and evaluation of an in-ear cardiac monitoring system, combining hardware prototyping, dataset creation, systematic machine-learning exploration, and embedded validation. The primary goal was to investigate the feasibility of reliable cardiac activity estimation from photoplethysmography (PPG) signals acquired inside the ear canal and to develop a transparent, reproducible framework for analysing and deploying machine learning (ML) models on embedded systems.

The results demonstrated that in-ear PPG sensing is a promising method for cardiac monitoring. Among the investigated sensor placements, the deep ear meatus consistently provided the highest signal quality under both static and dynamic conditions, followed by the concha, whereas the tragus and shallow meatus locations were more susceptible to motion artefacts. These findings, further confirmed by the resulting ML-model performances, demonstrate that the inner ear offers physiologically and mechanically favourable conditions for optical cardiac sensing, with the concha a practical alternative when deep insertion of the PPG sensor is not achievable, though with accuracy loss.

On the machine-learning side, systematic exploration of convolutional architectures revealed several key design insights. Batch normalisation and moderate pooling frequency consistently improved performance and convergence stability. Kernel size and dilation were shown to play important roles in determining temporal receptive fields, with moderate values (5–7 for kernel size and 2–4 for dilation) providing the best trade-off between performance and efficiency. For dilated architectures, the optimal configuration comprised five dilated layers with moderate bottleneck width, achieving the best balance between model complexity and accuracy. These findings contribute to a more structured understanding of how architectural choices affect model performance in the context of physiological signal analysis.

The embedded validation further confirmed that the conversion and deployment pipeline preserves model functionality, with numerical equivalence between the original model reference and its embedded conversion.

Although the final peak detection accuracy depends on the selected architecture and available hardware resources, the results demonstrate that high MCC values can be achieved when models are trained sufficiently and peak adjustment is applied. This indicates that the explored configurations are capable of providing reliable heart beat detection suitable for integration into a heart-rate estimation pipeline, with performance levels that can be matched to the constraints of the available computational capacity on the embedded device.

Several limitations should be acknowledged. The dataset was obtained from a single subject under controlled activity conditions, which limits the generalisability of the results. Broader validation across multiple participants and more diverse movement scenarios is needed to assess inter-subject variability and model robustness. Furthermore, the current measurement setup lacked full hardware synchronisation between PPG and ECG sensors, and embedded validation was performed on a small configuration space. Fully synchronised acquisition and direct on-device testing would improve temporal precision and provide more representative insights into real-time performance and energy efficiency.

In summary, this work provides a comprehensive and reproducible foundation for in-ear cardiac monitoring. Its contributions can be summarised as follows:

- A custom-fit, modular in-ear hardware system capable of multi-location PPG and acceleration measurements.
- A synchronised dataset with automated and manual alignment procedures for reliable cardiac reference generation.
- A configurable and open machine-learning framework that enables systematic exploration of CNN and Dilated CNN architectures.
- Empirical insights into model design trade-offs between accuracy, efficiency, and embedded feasibility.
- A validated end-to-end deployment pipeline from model training to embedded implementation.

Collectively, these contributions demonstrate the feasibility and potential of in-ear sensing as a robust platform for future health-monitoring wearables. Beyond the technical results, this work carries important societal implications. Among these, enabling non-invasive and continuous monitoring of cardiac activity in a compact and personally customisable form factor contributes to more accessible preventive healthcare and long-term physiological monitoring. The automatic model selection framework allows for optimal architecture selection under embedded constraints and supports personalised monitoring, as new subject-specific models can be trained when individual data is available, enabling more objective, transparent, and user-specific solutions. Together, these aspects promote more accurate and personalised on-device health analytics and lower the threshold for clinical and consumer adoption, representing a meaningful step toward accessible, precise, and personalised digital health technologies.

Bibliography

[1] C. Virginia Anikwe et al., "Mobile and wearable sensors for data-driven health monitoring system: State-of-the-art and future prospect," *Expert Systems with Applications*, vol. 202, p. 117 362, 2022, ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2022.117362. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095741742200714X.

[2] J. Dunn, R. Runge, and M. Snyder, "Wearables and the medical revolution," *Personalized Medicine*, vol. 15, no. 5, pp. 429–448, Sep. 2018. DOI: 10.2217/pme-2018-0044.

[3] A. Sapra, A. Malik, and P. Bhandari, *Vital sign assessment*, https://www-ncbi-nlm-nih-gov.tudelft.idm.oclc.org/books/NBK553213/, [Updated 2023 May 1]. In: StatPearls [Internet]. Treasure Island (FL): StatPearls Publishing; 2024 Jan-, 2024.

[4] A. K. Yetisen, J. L. Martinez-Hurtado, B. Ünal, A. Khademhosseini, and H. Butt, "Wearables in medicine," *Advanced Materials*, vol. 30, no. 33, p. 1 706 910, 2018. DOI: https://doi.org/10.1002/adma.201706910. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/adma.201706910. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/adma.201706910.

[5] P. Mao, H. Li, and Z. Yu, "A review of skin-wearable sensors for non-invasive health monitoring applications," *Sensors*, vol. 23, no. 7, 2023, ISSN: 1424-8220. DOI: 10.3390/s23073673. [Online]. Available: https://www.mdpi.com/1424-8220/23/7/3673.

[6] J.-Y. Choi et al., "Health-related indicators measured using earable devices: Systematic review," *JMIR Mhealth Uhealth*, vol. 10, no. 11, e36696, Nov. 2022, ISSN: 2291-5222. DOI: 10.2196/36696. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/36239201.

[7] E. Mejía-Mejía, J. M. May, R. Torres, and P. A. Kyriacou, "Pulse rate variability in cardiovascular health: A review on its applications and relationship with heart rate variability," *Physiological Measurement*, vol. 41, no. 7, 07TR01, 2020. DOI: 10.1088/1361-6579/ab998c.

[8] K. Fox et al., "Resting heart rate in cardiovascular disease," *JACC*, vol. 50, no. 9, pp. 823–830, 2007. DOI: 10.1016/j.jacc.2007.04.079. eprint: https://www.jacc.org/doi/pdf/10.1016/j.jacc.2007.04.079. [Online]. Available: https://www.jacc.org/doi/abs/10.1016/j.jacc.2007.04.079.

[9] C. K. H. Ne, J. Muzaffar, A. Amlani, and M. Bance, "Hearables, in-ear sensing devices for bio-signal acquisition: A narrative review," *Expert Review of Medical Devices*, vol. 18, no. sup1, pp. 95–128, 2021, PMID: 34904507. DOI: 10.1080/17434440.2021.2014321. eprint: https://doi.org/10.1080/17434440.2021.2014321. [Online]. Available: https://doi.org/10.1080/17434440.2021.2014321.

[10] M. Masè, A. Micarelli, and G. Strapazzon, "Hearables: New perspectives and pitfalls of in-ear devices for physiological monitoring. a scoping review," *Frontiers in Physiology*, vol. Volume 11 - 2020, 2020, ISSN: 1664-042X. DOI: 10.3389/fphys.2020.568886. [Online]. Available: https://www.frontiersin.org/journals/physiology/articles/10.3389/fphys.2020.568886.

[11] S. Ismail, U. Akram, and I. Siddiqi, "Heart rate tracking in photoplethysmography signals affected by motion artifacts: A review," *EURASIP Journal on Advances in Signal Processing*, vol. 2021, no. 1, p. 5, 2021, ISSN: 1687-6180. DOI: 10.1186/s13634-020-00714-2. [Online]. Available: https://doi.org/10.1186/s13634-020-00714-2.

[12] A. Ferlini, A. Montanari, C. Min, H. Li, U. Sassi, and F. Kawsar, "In-ear ppg for vital signs," *IEEE Pervasive Computing*, vol. 21, no. 1, pp. 65–74, 2022. DOI: 10.1109/MPRV.2021.3121171.

[13] M. Chustecki, "Benefits and risks of AI in health care: Narrative review," *Interactive Journal of Medical Research*, vol. 13, e53616, 2024. DOI: 10.2196/53616. [Online]. Available: https://www.i-jmr.org/2024/1/e53616.

[14] H. Habehh and S. Gohel, "Machine learning in healthcare," *Current Genomics*, vol. 22, no. 4, pp. 291–300, 2021, ISSN: 1875-5488. DOI: https://doi.org/10.2174/

1389202922666210705124359. [Online]. Available: https://www.benthamdirect.com/content/journals/cg/10.2174/1389202922666210705124359.

[15] G. Nie et al., *A review of deep learning methods for photoplethysmography data*, 2024. arXiv: 2401.12783 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2401.12783.

[16] J. Xu, Y. Zhang, M. Xie, W. Wang, and D. Zhu, "Real-time intelligent on-device monitoring of heart rate variability with ppg sensors," *Journal of Systems Architecture*, vol. 154, p. 103 240, 2024, ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2024.103240. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762124001772.

[17] N. Q. Mahardika T, Y. N. Fuadah, D. U. Jeong, and K. M. Lim, "Ppg signals-based blood-pressure estimation using grid search in hyperparameter optimization of cnn–lstm," *Diagnostics*, vol. 13, no. 15, 2023, ISSN: 2075-4418. DOI: 10.3390/diagnostics13152566. [Online]. Available: https://www.mdpi.com/2075-4418/13/15/2566.

[18] O. E. Gundersen, Y. Gil, and D. W. Aha, "On reproducible ai: Towards reproducible research, open science, and digital scholarship in ai publications," *AI Magazine*, vol. 39, no. 3, pp. 56–68, Sep. 2018. DOI: 10.1609/aimag.v39i3.2816. [Online]. Available: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2816.

[19] F. Bildirici, "Open-source ai: An approach to responsible artificial intelligence development," *Reflektif Sosyal Bilimler Dergisi*, pp. 73–81, 2024.

[20] *Dopple*, https://www.dopple.nl/, Accessed: 2025-10-11, Dopple, Inc., 2023.

[21] Natus Sensory, *Otoscan® – 3d ear scanning solution*, https://natussensory.com/products/otoscan/, Accessed: 2025-10-12.

[22] Dreve Otoplastik Shop, *Otoform®*, https://otoplastikshop.dreve.de/otoplastiken/catalog/category/view/s/otoformr/id/169/, Accessed: 2025-10-12, 2025.

[23] D. Ray, T. Collins, S. Woolley, and P. Ponnapalli, "A review of wearable Multi-Wavelength photoplethysmography," en, *IEEE Rev Biomed Eng*, vol. 16, pp. 136–151, Jan. 2023.

[24] B. A. Fallow, T. Tarumi, and H. Tanaka, "Influence of skin type and wavelength on light wave reflectance," *Journal of Clinical Monitoring and Computing*, vol. 27, no. 3, pp. 313–317, 2013, ISSN: 1573-2614. DOI: 10.1007/s10877-013-9436-7. [Online]. Available: https://doi.org/10.1007/s10877-013-9436-7.

[25] M. D. Peláez-Coca, A. Hernando, J. Lázaro, and E. Gil, "Impact of the ppg sampling rate in the pulse rate variability indices evaluating several fiducial points in different pulse waveforms," *IEEE Journal of Biomedical and Health Informatics*, vol. 26, no. 2, pp. 539–549, 2022. DOI: 10.1109/JBHI.2021.3099208.

[26] Renesas Electronics Corporation, *DA14695 - multi-core bluetooth 5.2 soc with system power management unit*, https://www.renesas.com/en/products/da14695, Accessed: 2025-10-12, 2025.

[27] Analog Devices, Inc., *Maxm86161 single-supply integrated optical module for hr and spo measurement - data sheet*, https://www.analog.com/media/en/technical-documentation/data-sheets/maxm86161.pdf, Rev. 1, 4/23. Accessed: 2025-10-01, 2023.

[28] Bosch Sensortec GmbH, *BMA580 - acceleration sensor*, https://www.bosch-sensortec.com/products/motion-sensors/accelerometers/bma580/, Accessed: 2025-10-12, 2025.

[29] M. Elgendi, "Optimal signal quality index for photoplethysmogram signals," en, *Bioengineering (Basel)*, vol. 3, no. 4, Sep. 2016.

[30] The SciPy community, *Scipy.stats.skew - scipy v1.16.2 manual*, https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.skew.html, Accessed: 2025-10-08, 2025.

[31] J. Wasilewski and L. Poloński, "An introduction to ecg interpretation," in *ECG Signal Processing, Classification and Interpretation: A Comprehensive Framework of Computational Intelligence*, A. Gacek and W. Pedrycz, Eds. London: Springer London, 2012, pp. 1–20, ISBN: 978-0-85729-868-3. DOI: 10.1007/978-0-85729-868-3_1. [Online]. Available: https://doi.org/10.1007/978-0-85729-868-3_1.

[32] J. Allen, "Photoplethysmography and its application in clinical physiological measurement," *Physiological Measurement*, vol. 28, R1–39, Apr. 2007. DOI: 10.1088/0967-3334/28/3/R01.

[33] Q. Zhu, X. Tian, C.-W. Wong, and M. Wu, "Learning your heart actions from pulse: Ecg waveform reconstruction from ppg," *IEEE Internet of Things Journal*, vol. 8, no. 23, pp. 16 734–16 748, 2021. DOI: 10.1109/JIOT.2021.3097946.

[34] S. Mohapatra, H. Palo, and M. Mohanty, "Detection of arrhythmia using neural network," Jan. 2018, pp. 97–100. DOI: 10.15439/2017KM42.

[35] Analog Devices. "Using reflectometry for a ppg waveform." Accessed: 2025-10-12, Analog Devices. [Online]. Available: https://www.analog.com/en/resources/design-notes/using-reflectometry-for-a-ppg-waveform.html.

[36] "Byteflies: Certified telemonitoring solutions for hospitals, clinical trials & research." Accessed: 2025-10-07, Byteflies. [Online]. Available: https://www.byteflies.com/.

[37] Byteflies NV, *Byteflies kit manual*, User manual, Byteflies, May 2020.

[38] The SciPy Community, *Scipy.signal.resample_poly - scipy v1.16.2 manual*, https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.resample_poly.html, Accessed: 2025-10-12, 2025.

[39] C. Xie, L. McCullum, A. Johnson, T. Pollard, B. Gow, and B. Moody, *Waveform database software package (wfdb) for python (version 4.1.0)*, https://doi.org/10.13026/9njx-6322, RRID:SCR_007345, 2023. DOI: 10.13026/9njx-6322.

[40] H. Tanaka, K. D. Monahan, and D. R. Seals, "Age-predicted maximal heart rate revisited," *JACC*, vol. 37, no. 1, pp. 153–156, 2001. DOI: 10.1016/S0735-1097(00)01054-8. eprint: https://www.jacc.org/doi/pdf/10.1016/S0735-1097%2800%2901054-8. [Online]. Available: https://www.jacc.org/doi/abs/10.1016/S0735-1097%2800%2901054-8.

[41] M. Jorda, P. Valero-Lara, and A. J. Pena, "Performance evaluation of cudnn convolution algorithms on nvidia volta gpus," *IEEE Access*, vol. 7, no. 8721631, pp. 70 461–70 473, 2019. DOI: 10.1109/ACCESS.2019.2918851.

[42] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, 2015. arXiv: 1502.03167 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1502.03167.

[43] PyTorch Team, *Torch.nn.batchnorm1d - pytorch documentation*, https://docs.pytorch.org/docs/stable/generated/torch.nn.BatchNorm1d.html, Accessed: 2025-10-04, 2025.

[44] H. Gholamalinezhad and H. Khosravi, *Pooling methods in deep neural networks, a review*, 2020. arXiv: 2009.07485 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2009.07485.

[45] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, 2015. arXiv: 1505.04597 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1505.04597.

[46] I. Loshchilov and F. Hutter, *Decoupled weight decay regularization*, 2019. arXiv: 1711.05101 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1711.05101.

[47] *Bcewithlogitsloss - pytorch documentation*, PyTorch. Accessed: Sep. 30, 2025. [Online]. Available: https://docs.pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html.

[48] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, *Focal loss for dense object detection*, 2018. arXiv: 1708.02002 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1708.02002.

[49] J. Hosang, R. Benenson, and B. Schiele, *Learning non-maximum suppression*, 2017. arXiv: 1705.02950 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1705.02950.

[50] D. Chicco and G. Jurman, "The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation," *BMC Genomics*, vol. 21, no. 1, p. 6, 2020, ISSN: 1471-2164. DOI: 10.1186/s12864-019-6413-7. [Online]. Available: https://doi.org/10.1186/s12864-019-6413-7.

[51] SciPy Community, *Scipy.optimize.linear_sum_assignment*, https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linear_sum_assignment.html, Accessed: 2025-10-13, SciPy version 1.16.2, 2025.

[52] D. F. Crouse, "On implementing 2d rectangular assignment algorithms," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 52, no. 4, pp. 1679–1696, 2016. DOI: 10.1109/TAES.2016.140952.

[53] D. A. Van Veldhuizen, G. B. Lamont, et al., "Evolutionary computation and convergence to a pareto front," in *Late breaking papers at the genetic programming 1998 conference*, 1998, pp. 221–228.

[54] STMicroelectronics, *Stm32h755zi – high-performance arm® cortex®-m7 + m4 mcu*, Accessed: 2025-11-30, 2025. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32h755zi.html#overview.

[55] STMicroelectronics, *Nucleo-h755zi-q – stm32 nucleo-144 development board*, Accessed: 2025-11-30, 2025. [Online]. Available: https://www.st.com/en/evaluation-tools/nucleo-h755zi-q.html.

[56] Google AI Edge / Google AI for Developers, *Litert overview*, https://ai.google.dev/edge/litert, Last updated 2025-05-19 UTC, 2025. [Online]. Available: https://ai.google.dev/edge/litert.

[57] google-ai-edge, *Ai edge torch: Supporting pytorch models with the google ai edge tflite runtime*, https://github.com/google-ai-edge/ai-edge-torch, Accessed: 2025-09-30, 2025.

[58] STMicroelectronics, *Stm32cube.ai*, https://stm32ai.st.com/stm32-cube-ai/, Accessed: 2025-10-14, 2025.

[59] STMicroelectronics, *Evaluation report and metrics — st edge ai core technology 2.2.0*, https://stedgeai-dc.st.com/assets/embedded-docs/evaluation_metrics.html, Accessed: 2025-10-29, 2025.

[60] A. Ltd., *Cortex-m0 technical reference manual*, Rev. 3, Document ID: ARM DDI 0432I, 30 Nov 2009, Cambridge, UK, Nov. 2009. [Online]. Available: https://documentation-service.arm.com/static/5e8e294afd977155116a6a5b.

Algorithms A

This section contains the algorithms for building the different ML-models.

A.1 Dynamic Encoder-Decoder CNN

Algorithm 1 Dynamic encoder-decoder CNN construction

```
Require: Input signal $x^{(0)} \in \mathbb{R}^{1\times T}$
                                                        ▷ Configuration parameters:
 1: $B$: number of convolutional blocks in the encoder
 2: $k$: convolutional kernel size
 3: $d$: dilation factor controlling kernel spacing
 4: pool_every: number of blocks after which pooling is applied
 5: double_every: number of blocks after which channels are doubled
 6: $C_{\text{base}}$: base number of channels in the first block
 7: $f_{\text{pool}}$: pooling function (max, avg, etc.)
 8: $f_{\text{act}}$: activation function ($SiLU$ or $ReLU$)
                                                        ▷ Initialization:
 1: $C_0 \leftarrow 1$          ▷ initial number of channels (input has one channel)
 2: $T_0 \leftarrow T$          ▷ initial temporal length (outlined for clarity)
 3: upscaler_count $\leftarrow 0$
                                                        ▷ Encoder:
 4: for $i = 1$ to $B$ do
 5:     $C_i \leftarrow C_{\text{base}} \times 2^{\lfloor (i-1)/\text{double\_every} \rfloor}$     ▷ double channels
 6:     $C_{\text{in}} \leftarrow C_{i-1}$, $C_{\text{out}} \leftarrow C_i$     ▷ Set input and output channels
 7:     $z^{(i)} \leftarrow \text{Conv1D}(x^{(i-1)}, C_{\text{in}}, C_{\text{out}}, k, d)$     ▷ Eq. 2.16
 8:     if Batch Normalisation enabled then
 9:         $z^{(i)} \leftarrow \text{BatchNorm}(z^{(i)})$     ▷ Eq. 2.17
10:     end if
11:     $x^{(i)} \leftarrow f_{\text{act}}(z^{(i)})$     ▷ apply non-linearity, Eq. 2.18 or Eq. 2.20
12:     if $i$ mod pool_every $= 0$ then
13:         $x^{(i)} \leftarrow f_{\text{pool}}(x^{(i)}, p = 2)$     ▷ Eqs. 2.22–2.24
14:         $T_i \leftarrow T_i/2$
15:         upscaler_count $\leftarrow$ upscaler_count $+ 1$
16:     end if
17: end for
                                                        ▷ Decoder:
18: if upscaler_count $> 0$ then
19:     $x^{(B+1)} \leftarrow \text{Upsample}(x^{(B)}, s = 2^{\text{upscaler\_count}})$     ▷ Eq. 2.25
20: else
21:     $x^{(B+1)} \leftarrow x^{(B)}$
22: end if
23: $\hat{y} \leftarrow \text{Conv1D}(x^{(B+1)}, C^{(B)}, C_{\text{out}} = 1, k = 1, d = 0)$     ▷ final 1×1 projection
24: return $\hat{y} \in \mathbb{R}^{1\times T}$     ▷ predicted peak-likelihood signal
```

A.2 Dilated Encoder-Decoder CNN

Algorithm 2 Dilated Encoder-Decoder CNN construction (Consolidated Config)

```
Require: Input signal $x^{(0)} \in \mathbb{R}^{C_{\text{in}} \times T}$
                                                      ▷ Configuration parameters:
 1: $C_{enc1}, C_{enc2}, C_b, C_{dec1}, C_{dec2}, C_{\text{out}}$: Channels for encoder, bottleneck, decoder, and output
 2: $k_{enc1}, k_{enc2}, k_d, k_{dec1}, k_{dec2}$: Kernel sizes for encoder, dilated bottleneck, and decoder layers
 3: $N_d$: number of layers in the bottleneck
 4: $d_{\text{base}}$: base dilation factor
 5: $f_{\text{pool}}$: pooling function (max, avg, etc.)
 6: $f_{\text{act}}$: activation function ($SiLU$ or $ReLU$)

    procedure ConvBlock($x_{\text{in}}$, $C_{\text{in}}$, $C_{\text{out}}$, $k$, $d$, $f_{\text{act}}$)
 1: $p \leftarrow \lfloor d \cdot (k-1)/2 \rfloor$            ▷ Calculate padding to maintain size
 2: $z \leftarrow \text{Conv1D}(x_{\text{in}}, C_{\text{in}}, C_{\text{out}}, k, d, p)$      ▷ Eq. 2.16
 3: $z \leftarrow \text{BatchNorm}(z)$                          ▷ Eq. 2.17
 4: return $f_{\text{act}}(z)$                                  ▷ Eq. 2.18 or 2.20
                                                      ▷ Encoder:
 5: $x^{(1)} \leftarrow \text{ConvBlock}(x^{(0)}, C_{\text{in}}, C_{enc1}, k_{enc1}, d=1, f_{\text{act}})$
 6: $x^{(1,\text{pool})} \leftarrow f_{\text{pool}}(x^{(1)}, p=2)$                ▷ Eqs. 2.22–2.24
 7: $x^{(2)} \leftarrow \text{ConvBlock}(x^{(1,\text{pool})}, C_{enc1}, C_{enc2}, k_{enc2}, d=1, f_{\text{act}})$
 8: $x^{(2,\text{pool})} \leftarrow f_{\text{pool}}(x^{(2)}, p=2)$                ▷ Eqs. 2.22–2.24
                                                      ▷ Bottleneck (Dilated Convolutions):
 9: $z^{(0)} \leftarrow x^{(2,\text{pool})}$
10: for $j = 0$ to $N_d - 1$ do
11:     $C_{\text{prev}} \leftarrow C_{enc2}$ if $j = 0$ else $C_b$
12:     $d_j \leftarrow d_{\text{base}}^j$                      ▷ Eq. 2.26
13:     $z^{(j+1)} \leftarrow \text{ConvBlock}(z^{(j)}, C_{\text{prev}}, C_b, k_d, d_j, f_{\text{act}})$
14: end for
                                                      ▷ Decoder:
15: $y^{(1)} \leftarrow \text{Upsample}(z^{(N_d)}, s=2)$                  ▷ Eq. 2.25
16: $y^{(1,\text{conv})} \leftarrow \text{ConvBlock}(y^{(1)}, C_b, C_{dec1}, k_{dec1}, d=1, f_{\text{act}})$
17: $y^{(2)} \leftarrow \text{Upsample}(y^{(1,\text{conv})}, s=2)$               ▷ Eq. 2.25
18: $y^{(2,\text{conv})} \leftarrow \text{ConvBlock}(y^{(2)}, C_{dec1}, C_{dec2}, k_{dec2}, d=1, f_{\text{act}})$
                                                      ▷ Output Projection:
19: $\hat{y} \leftarrow \text{Conv1D}(y^{(2,\text{conv})}, C_{dec2}, C_{\text{out}}, k=1)$     ▷ Final 1×1 projection
20: return $\hat{y} \in \mathbb{R}^{C_{\text{out}} \times T}$
```

Search Space Configurations B

This chapter contains all the different search space configurations used to create results and graphs.

B.1 Standard CNN

Table B.1: Standard CNN configuration grids used throughout the single-location analysis. Columns list the values explored per experiment. Parameters not listed under a column follow the common settings detailed in the notes. Reported search-space sizes exclude configurations whose model size exceeded 2 MB.

Parameter	**BatchNorm**	**Pooling**	**Channels**	**Convolution**
base_channels	{8,16}	{8,16}	{8,16,32,64}	{8,16}
num_blocks	{3,5}	{3,5}	{3,5}	{3,5}
kernel_size	{3,5}	{3,5}	{3,5}	{3,5,7,9}
dilation	{1,2}	{1,2}	{1,2,5}	{1,2,4,8}
use_batchnorm	{True, False}	True	True	True
pool_every	{1,2}	{1,2,4,8}	{2,4}	{2,4}
pooling	max	{max, avg, lp_2}	max	max
double_every	{1,2}	{1,2,4,8}	{1,2,4,8}	{2,4}
activation	silu	silu	silu	silu
Search space	32	768	384 → 360 (<2MB)	256

Parameter	**Activation**
base_channels	{8,16,32}
num_blocks	{3,5}
kernel_size	{3,5,7,9}
dilation	{2,4}
use_batchnorm	True
pool_every	{2,4}
pooling	max
double_every	{2,4}
activation	{silu, relu}
Search space	384

Common settings:
tune_threshold=enabled, NMS=disabled, event toleration=disabled, epochs=8, loss=BCE

B.2 Dilated

Table B.2: Dilated CNN configuration grids. Columns list the values explored per experiment. Parameters not listed under a column follow the common settings detailed in the notes. Reported search-space sizes exclude configurations whose model size exceeded 2 MB.

Parameter	**Bottleneck / Dilation**	**Conv Block**	**Encoder**	**Decoder**
`enc1_channels`	{8,16}	{8,16}	{8,16,32}	{8,16}
`enc1_kernel`	5	5	{3,5,7}	5
`enc2_channels`	{8,16,32}	{8,16}	{8,16,32}	16
`enc2_kernel`	3	3	{3,5,7}	5
`bottleneck_channels`	{16,32,64,128}	{16,32}	{16,32}	{32,64}
`num_dilated_layers`	{3,5,7}	5	5	{3,5}
`dilated_kernel`	{3,5}	{3,5,7,9}	{3,5}	{3,5}
`base_dilation`	{2,3}	{1,2,3,4}	2	2
`dec1_kernel`	5	5	5	{3,5,7}
`dec2_kernel`	5	5	5	{3,5,7}
Search space	*288 → 282 (<2MB)*	*128*	*128*	*144*

Parameter	**Similarity**
`enc1_channels`	{8,16}
`enc1_kernel`	{3,5}
`enc2_channels`	{16,32}
`enc2_kernel`	{3,5}
`dec1_channels`	{16,32}
`dec1_kernel`	{3,5}
`dec2_channels`	{8,16}
`dec2_kernel`	{3,5}
`bottleneck_channels`	32
`num_dilated_layers`	5
`dilated_kernel`	3
`base_dilation`	2
`activation`	`silu`
`pooling`	`avg`
Search space	96

Common settings:
`tune_threshold`=enabled, `NMS`=disabled, `event toleration`=disabled, `epochs`=8, `pooling`=max, `use_batchnorm`=True, `loss`=BCE

Unless otherwise specified:
`enc1_channels` = `dec2_channels`, `enc2_channels` = `dec1_channels`

B.3 Pre and post-processing

Table B.3: Configuration overview for pre-/post-processing analysis, comparative analysis across sensor locations, and embedded validation.

Standard CNN			
Parameter	**Pre/Post**	**Comparative**	**Embedded**
`base_channels`	{8,16}	{8,16,32}	16
`num_blocks`	{3,5}	{3,5}	5
`kernel_size`	{5,7}	{5,7}	7
`dilation`	2	2	2
`pool_every`	{2,4}	{2,4}	2
`double_every`	{2,4}	2	2
`activation`	`silu`	{`silu`,`relu`}	`silu`
Total Configurations	32	48	1

Complex Dilated CNN			
Parameter	**Pre/Post**	**Comparative**	**Embedded**
`enc1_channels`	8	{8,16}	16
`enc1_kernel`	{3,5}	5	5
`enc2_channels`	16	{8,16}	16
`enc2_kernel`	{3,5}	5	5
`bottleneck_channels`	{16,32}	{8,16,32}	64
`num_dilated_layers`	5	5	5
`dilated_kernel`	{3,5}	{3,5}	5
`base_dilation`	2	2	2
`activation`	`silu`	{`silu`,`relu`}	`silu`
Total Configurations	16	48	1

Common settings:
`tune_threshold`=enabled, `NMS`=disabled, `event toleration`=disabled, `epochs`=8, `pooling`=max, `use_batchnorm`=True, `loss`=BCE

C Figures

C.1 ECG Filtering

This section presents Figures C.1–C.2, which illustrate the effects of ECG filtering.

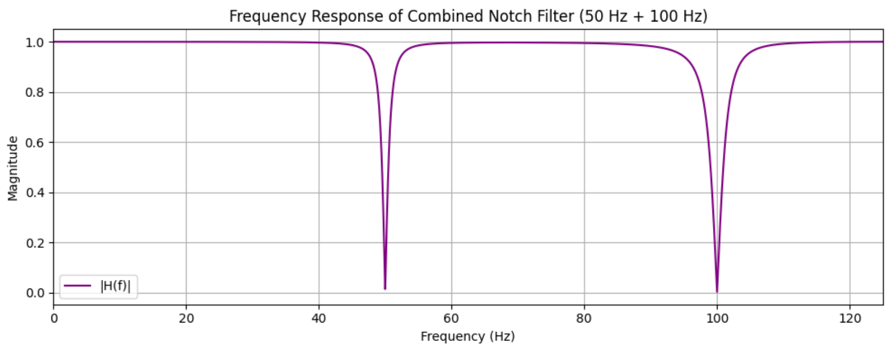


Figure C.1: Frequency response of the cascaded 50 Hz and 100 Hz IIR notch filters.

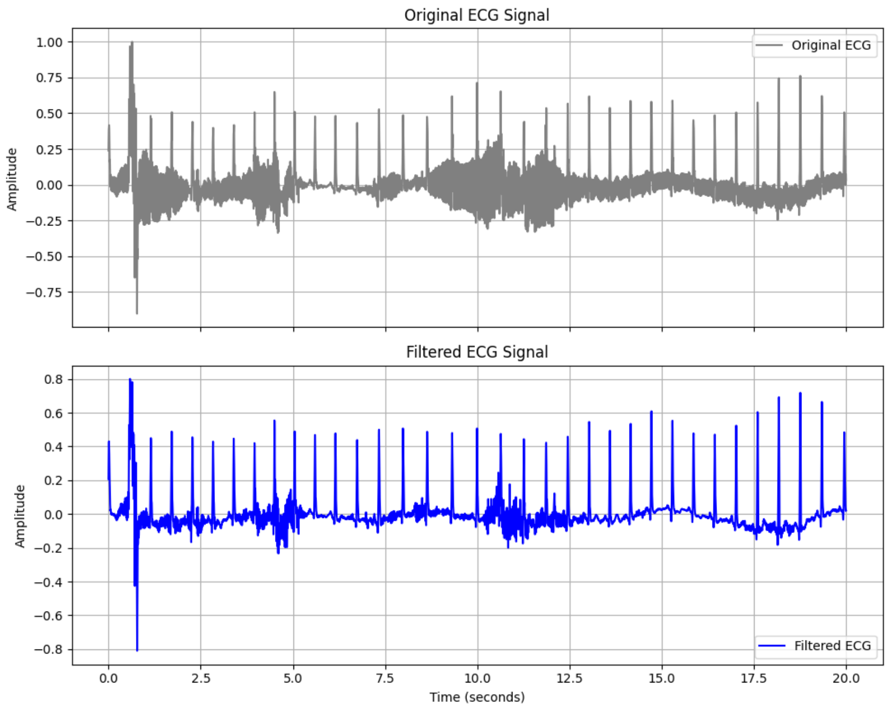


Figure C.2: Comparison between the original ECG signal and the filtered ECG signal.

C.2 Pooling strategies

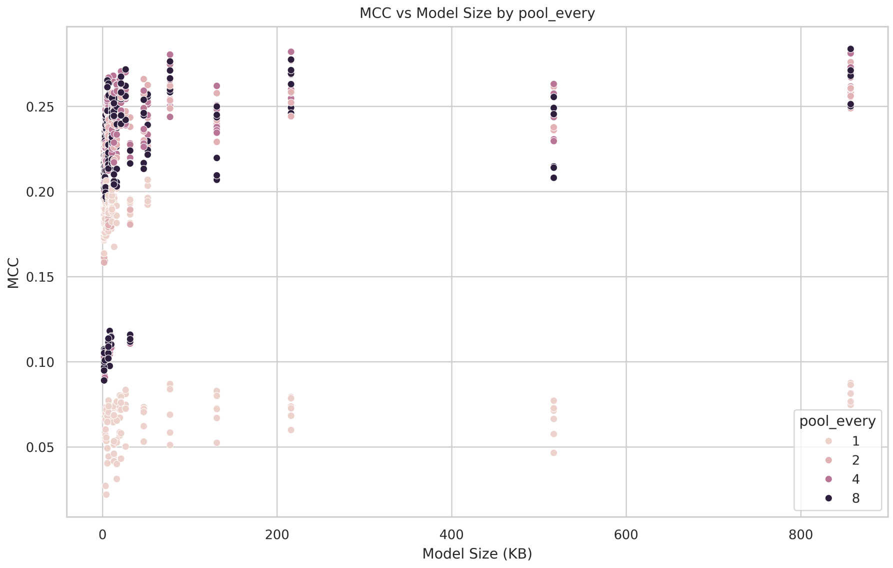
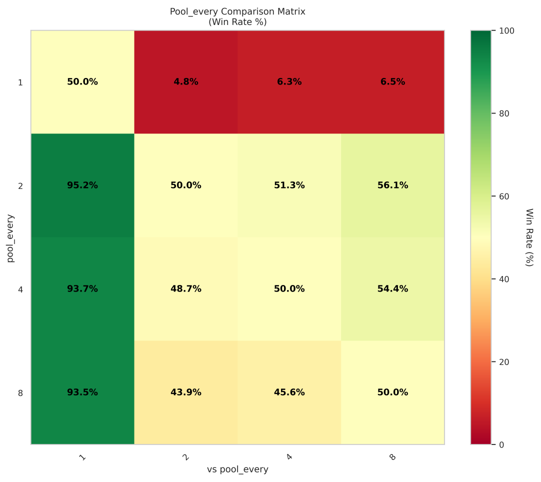
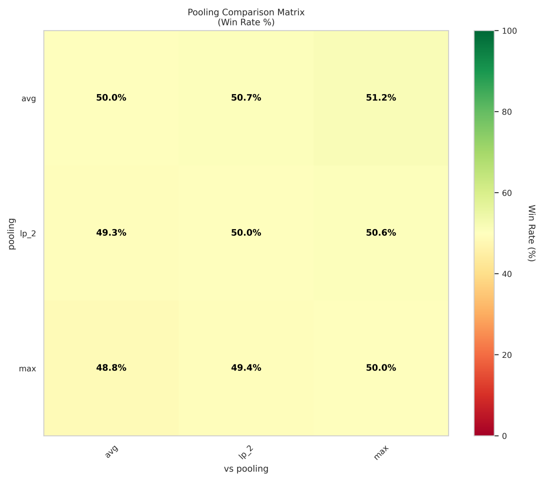


Figure C.3: Relationship between model size (KB) and MCC for different pool_every values in CNN configurations.



	1	2	4	8
1	50.0%	4.8%	6.3%	6.5%
2	95.2%	50.0%	51.3%	56.1%
4	93.7%	48.7%	50.0%	54.4%
8	93.5%	43.9%	45.6%	50.0%

(a) Win rate matrix for different `pool_every` configurations.



	avg	lp_2	max
avg	50.0%	50.7%	51.2%
lp_2	49.3%	50.0%	50.6%
max	48.8%	49.4%	50.0%

(b) Win rate matrix for different pooling types.

Figure C.4: Win rate matrices for pooling-related parameters in CNN configurations.

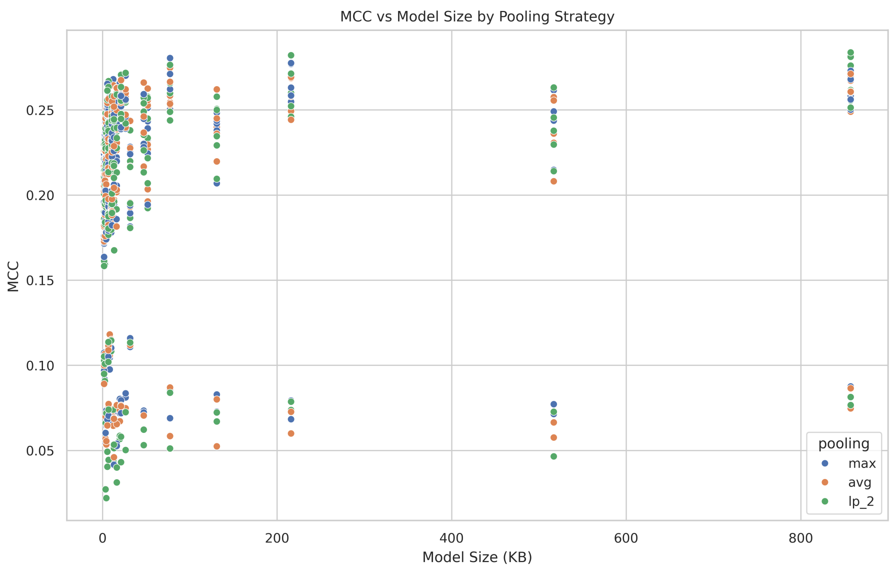
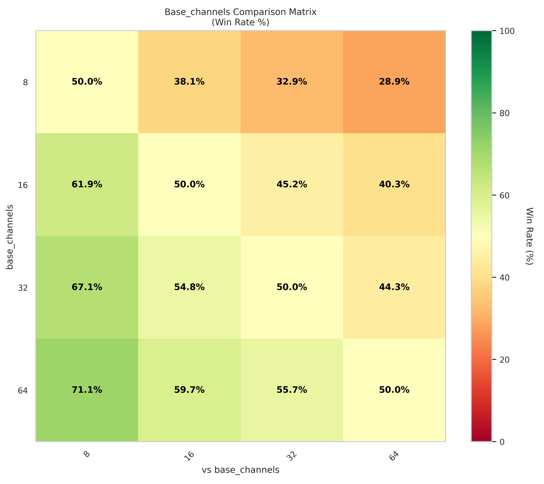
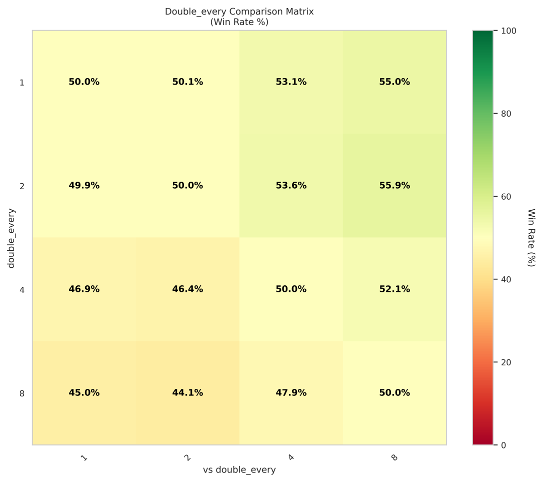


Figure C.5: Relationship between model size (KB) and MCC for different pooling types in CNN configurations.

C.3 Standard CNN channels

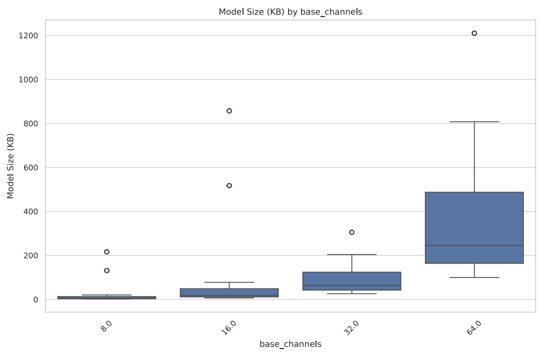


(a) Win rate matrix for different `base_channel` values.

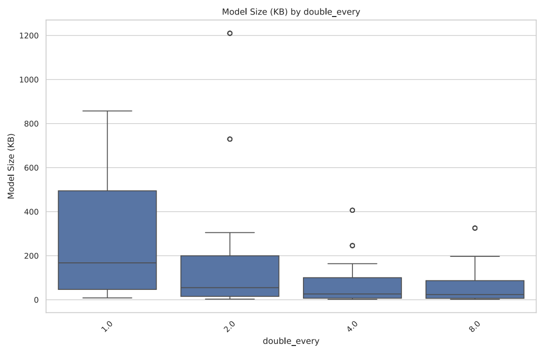


(b) Win rate matrix for different `double_every` values.

Figure C.6: Win rate matrices for channel-related parameters in the standard CNN encoder.



(a) Box plot showing the distribution of model sizes (KB) for different `base_channel` values.



(b) Box plot showing the distribution of model sizes (KB) for different `double_every` values.

Figure C.7: Box plots of model size distributions for different `base_channel` and `double_every` configurations.

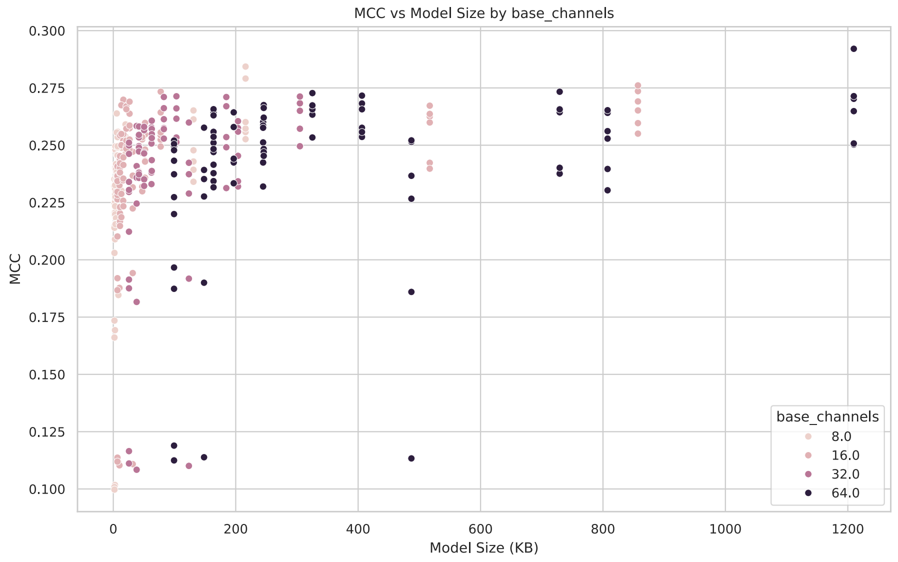


Figure C.8: Relationship between model size (KB) and MCC for different `base_channel` values in CNN configurations.

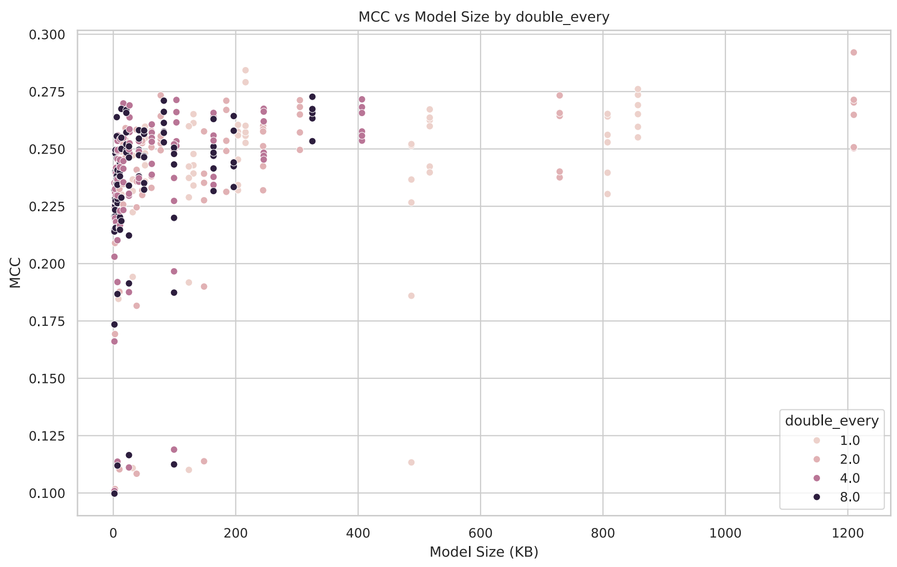
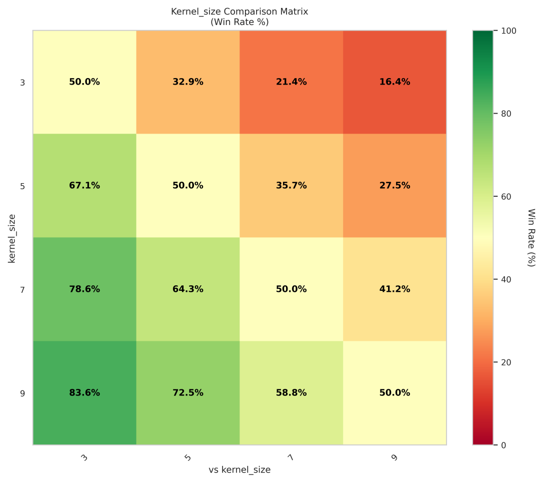
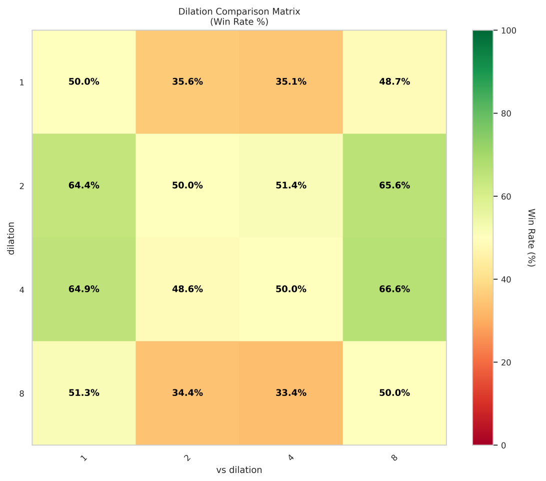


Figure C.9: Relationship between model size (KB) and MCC for different `double_every` values in CNN configurations.

C.4 Standard convolution layer



(a) Win rate matrix comparing different `kernel_size` values on MCC performance.



(b) Win rate matrix comparing different `dilation` values on MCC performance.

Figure C.10: Win rate matrices for different `kernel_size` and `dilation` parameters.

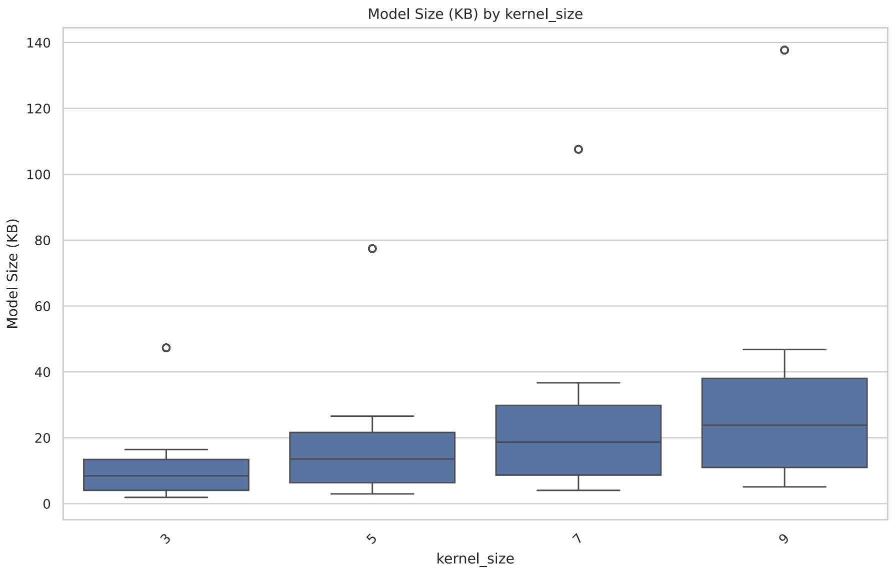


Figure C.11: Box plot showing the distribution of model sizes (KB) for different `kernel_size` values

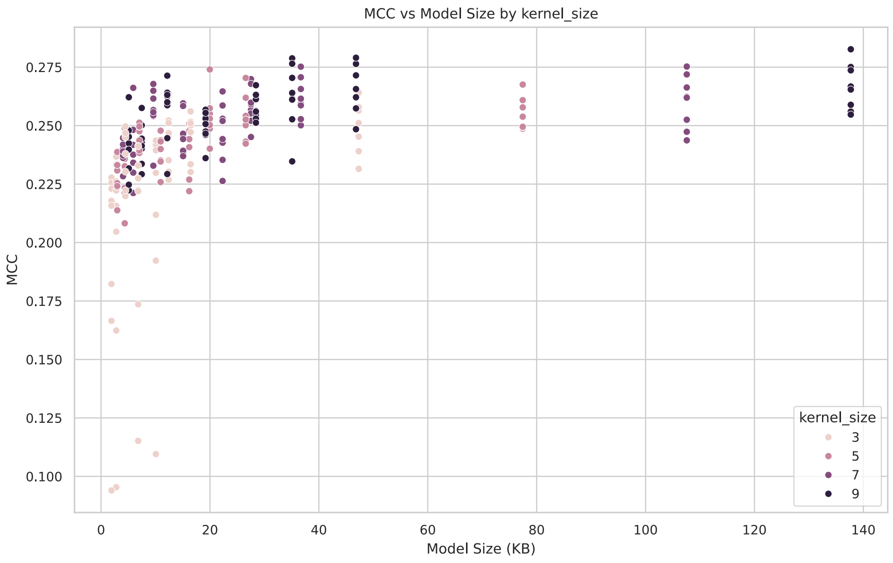


Figure C.12: Relationship between model size (KB) and MCC for different `kernel_size` values in standard CNN configurations.

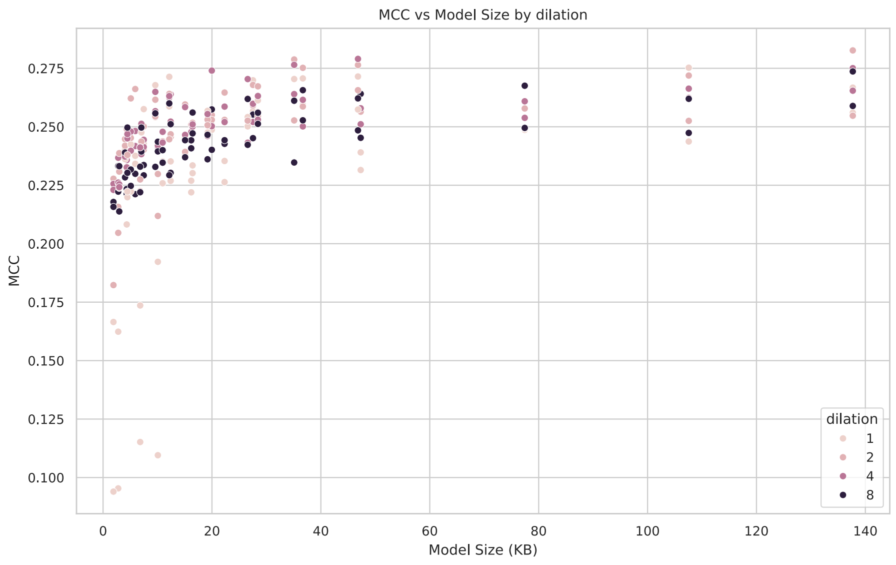


Figure C.13: Relationship between model size (KB) and MCC for different dilation values in standard CNN configurations.

C.5 Dilated bottleneck stage

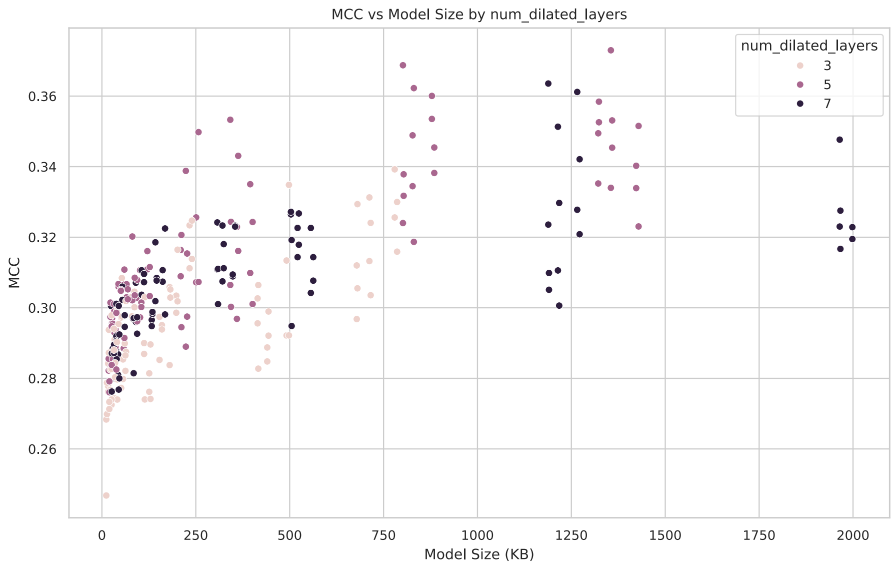


Figure C.14: Relationship between model size (KB) and MCC for different number of dilation layers in the Dilated CNN configurations.

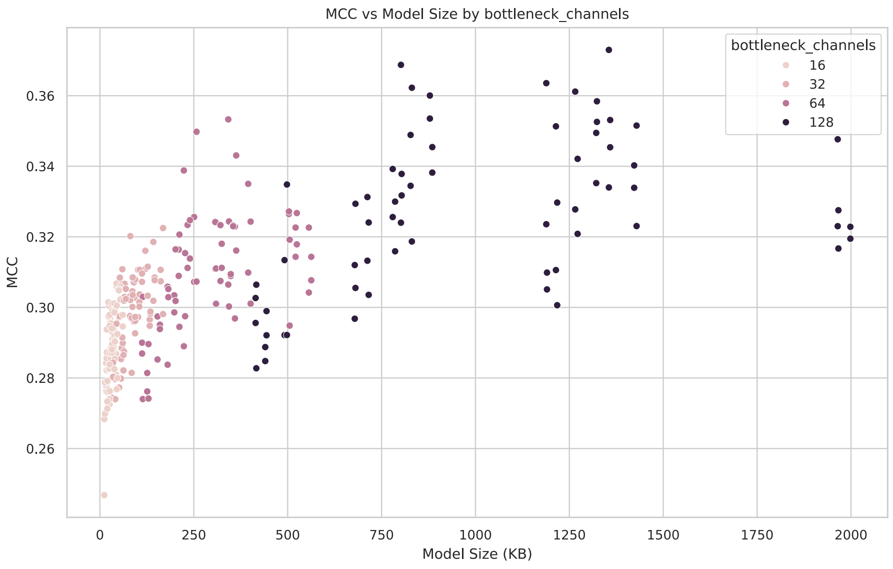
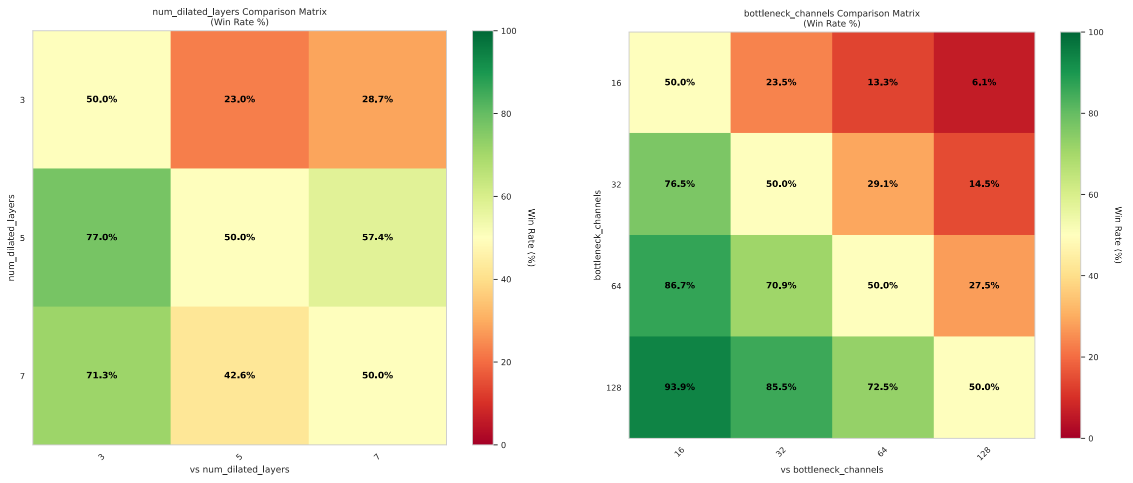


Figure C.15: Relationship between model size (KB) and MCC for different number of bottleneck channels in the Dilated CNN configurations.



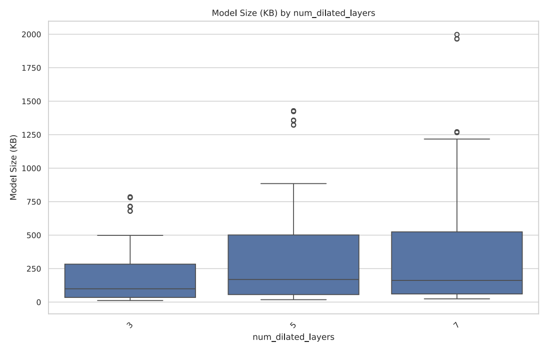
	3	5	7
3	50.0%	23.0%	28.7%
5	77.0%	50.0%	57.4%
7	71.3%	42.6%	50.0%

	16	32	64	128
16	50.0%	23.5%	13.3%	6.1%
32	76.5%	50.0%	29.1%	14.5%
64	86.7%	70.9%	50.0%	27.5%
128	93.9%	85.5%	72.5%	50.0%

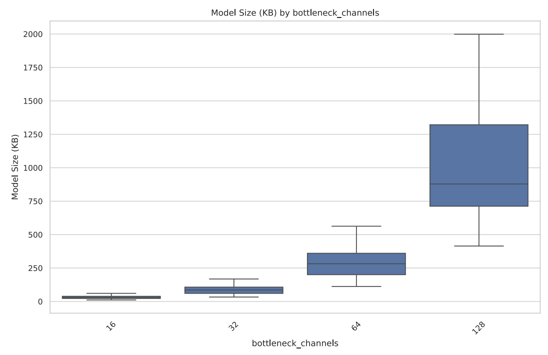
(a) Win rate matrix comparing the number of dilated layers on MCC performance.

(b) Win rate matrix comparing the number of bottleneck channels on MCC performance.

Figure C.16: Win rate matrices for dilated CNN parameters: dilated layers and bottleneck channels.



(a) Box plot illustrating the distribution of model sizes as a function of the number of dilated layers.



(b) Box plot illustrating the distribution of model sizes as a function of the number of bottleneck channels.

Figure C.17: Box plots of model size distributions for dilated CNN configurations: number of dilated layers and number of bottleneck channels.

C.6 Dilated convolution layer

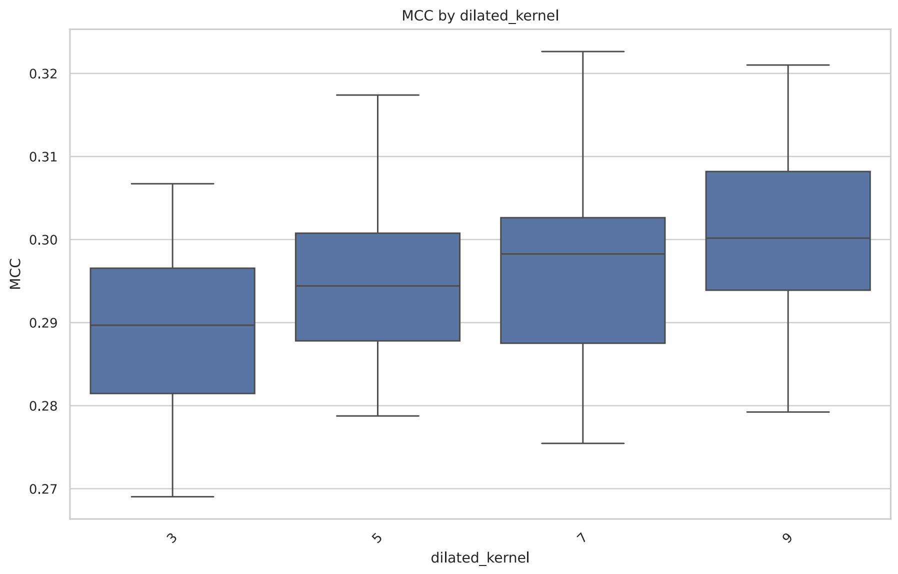
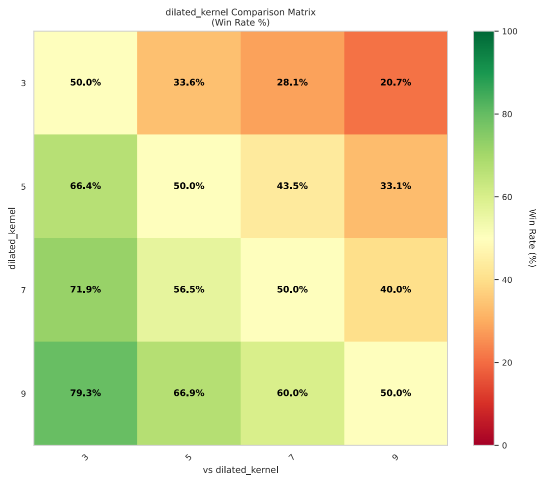
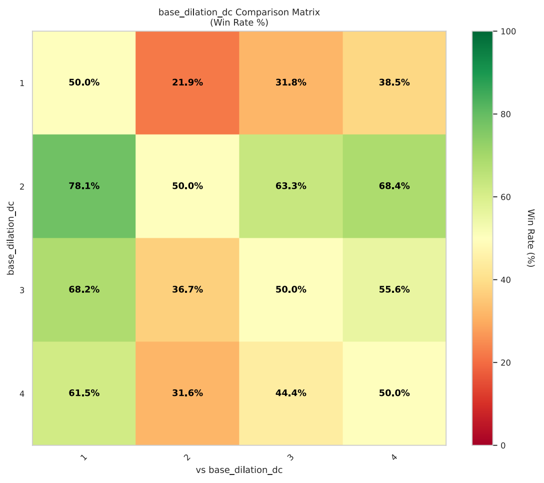


Figure C.18: Box plot showing the distribution of MCC scores for different kernel window sizes across 128 configurations.



	3	5	7	9
3	50.0%	33.6%	28.1%	20.7%
5	66.4%	50.0%	43.5%	33.1%
7	71.9%	56.5%	50.0%	40.0%
9	79.3%	66.9%	60.0%	50.0%

(a) Win-rate matrix comparing different kernel window sizes in the Dilated CNN.



	1	2	3	4
1	50.0%	21.9%	31.8%	38.5%
2	78.1%	50.0%	63.3%	68.4%
3	68.2%	36.7%	50.0%	55.6%
4	61.5%	31.6%	44.4%	50.0%

(b) Win-rate matrix comparing different base dilation factors in the Dilated CNN.

Figure C.19: Win-rate matrices for kernel window size and base dilation parameters in the Dilated CNN.

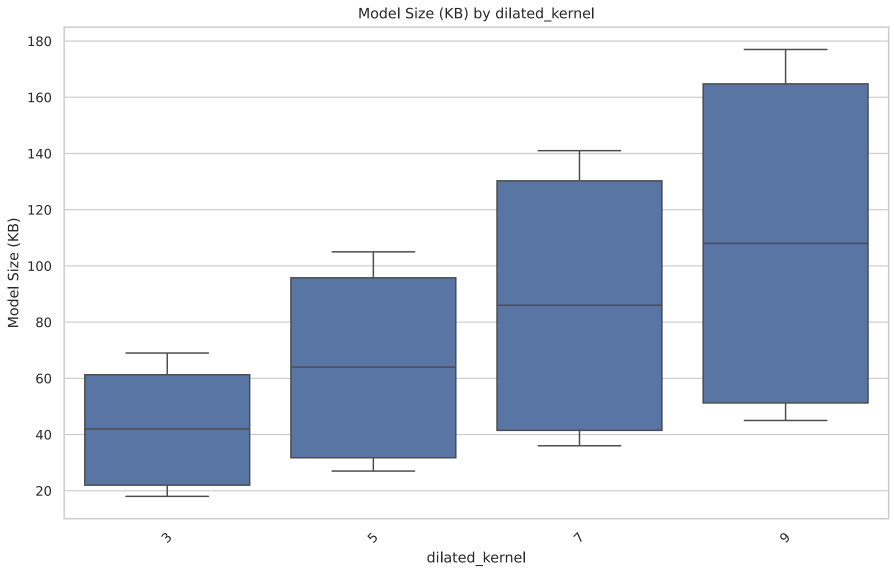


Figure C.20: Box plot illustrating the distribution of model sizes as a function of the kernel window size.

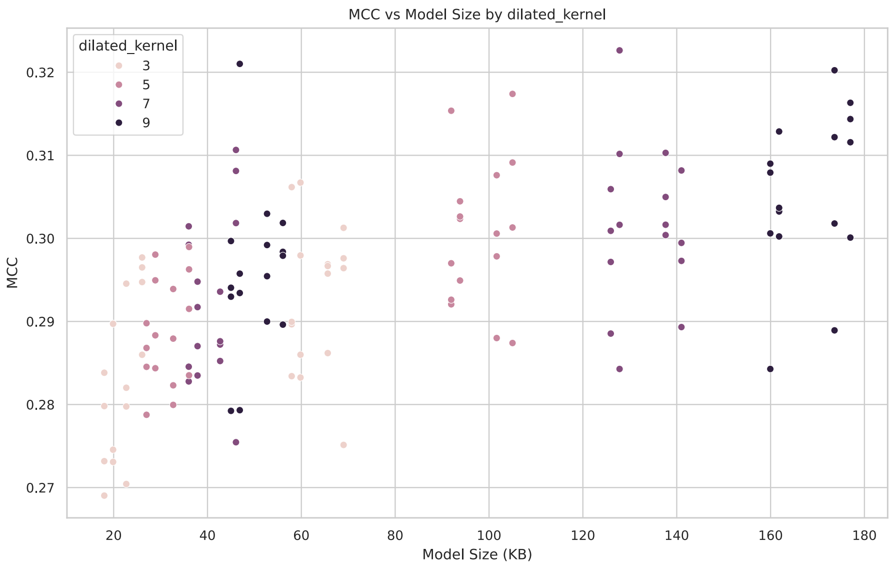


Figure C.21: Relationship between model size and MCC for different kernel window sizes in the Dilated CNN.

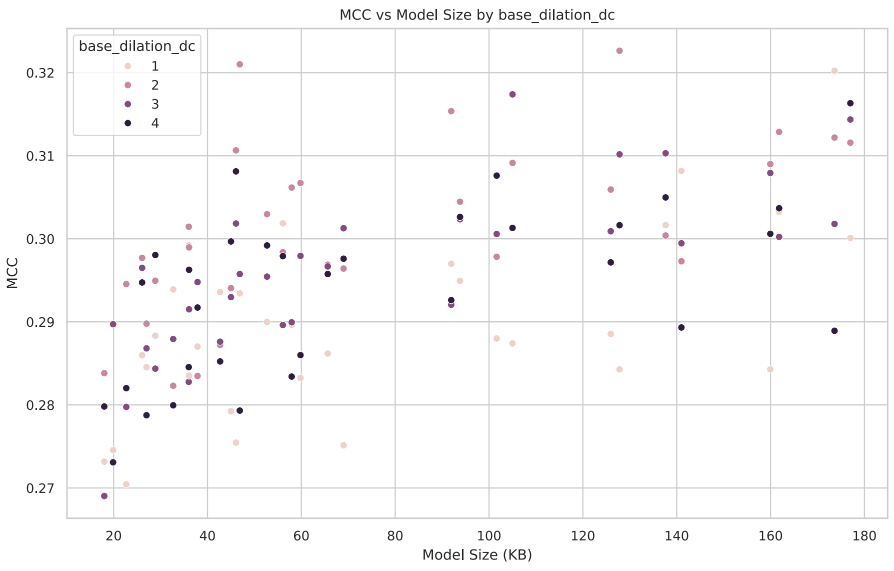
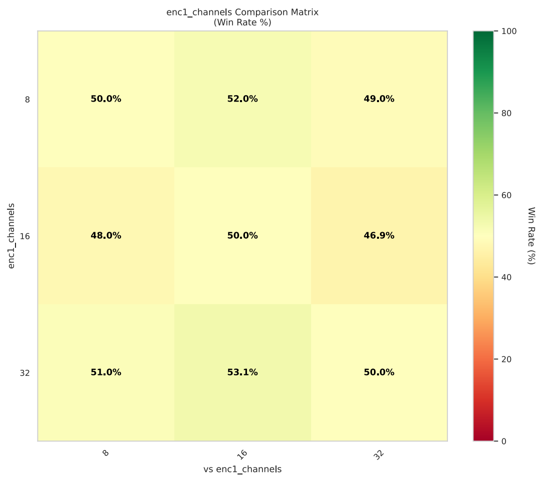
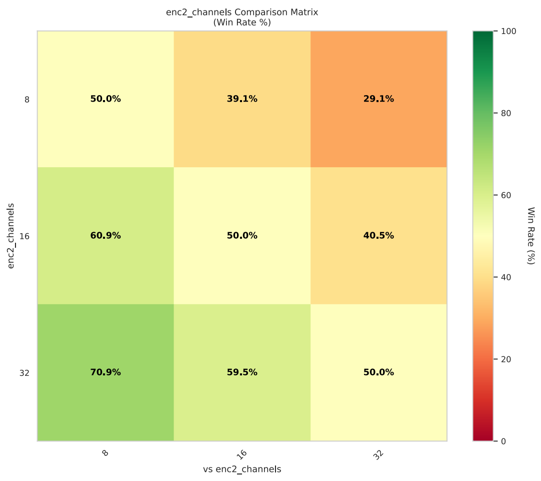


Figure C.22: Relationship between model size and MCC for different base dilation values in the Dilated CNN.

C.7 Dilated encoder architecture



(a) Win-rate matrix comparing different `enc1_channels` values.



(b) Win-rate matrix comparing different `enc2_channels` values.

Figure C.23: Win-rate matrices for varying `enc1_channels` and `enc2_channels` in the Dilated CNN architecture.

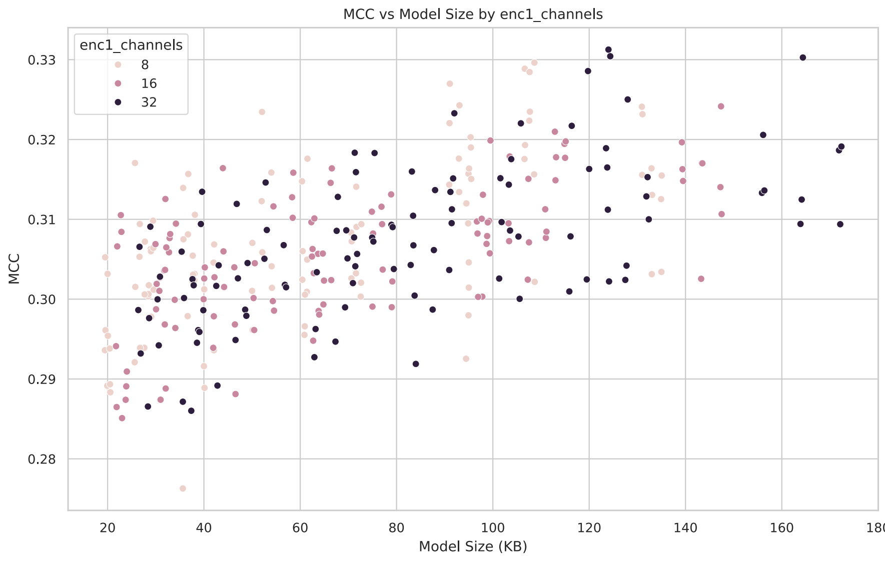


Figure C.24: Relationship between model size and MCC for different `enc1_channels` values in the Dilated CNN.

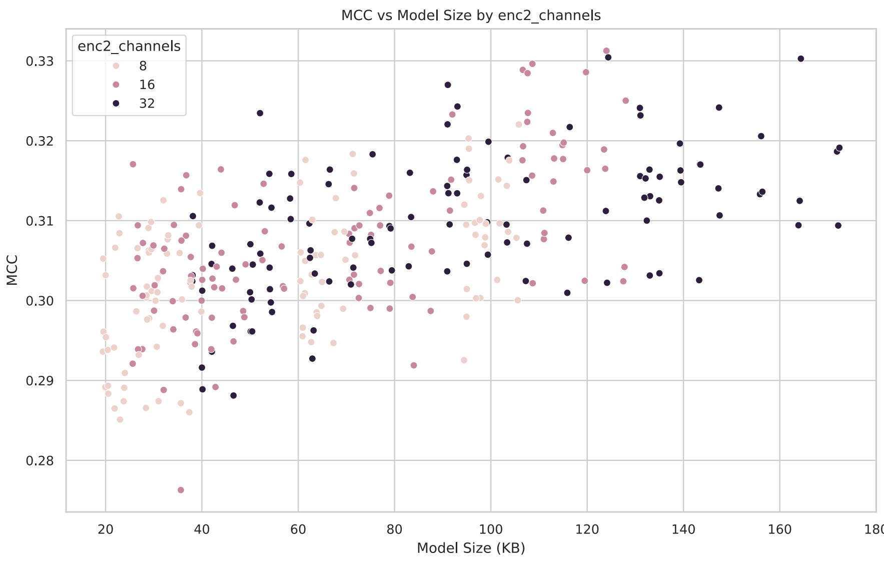
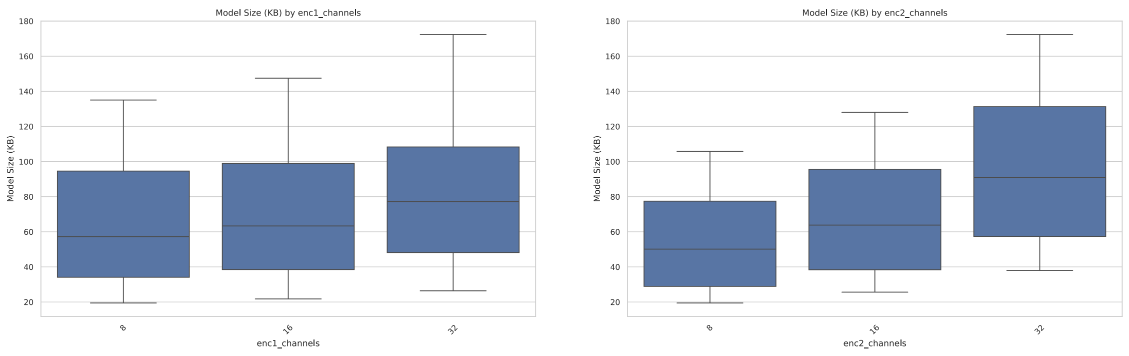


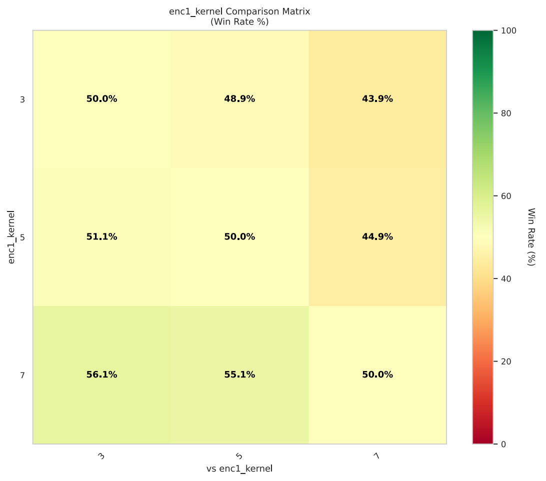
Figure C.25: Relationship between model size and MCC for different `enc2_channels` values in the Dilated CNN.



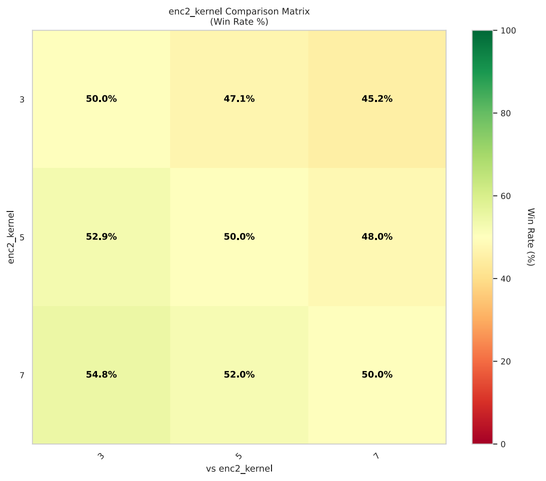
(a) Distribution of model sizes for different `enc1_channels` values.

(b) Distribution of model sizes for different `enc2_channels` values.

Figure C.26: Model size distributions for varying `enc1_channels` and `enc2_channels` values.



(a) Win-rate matrix comparing different `enc1_kernel` values.



(b) Win-rate matrix comparing different `enc2_kernel` values.

Figure C.27: Win-rate matrices for varying `enc1_kernel` and `enc2_kernel` values in the Dilated CNN encoder.

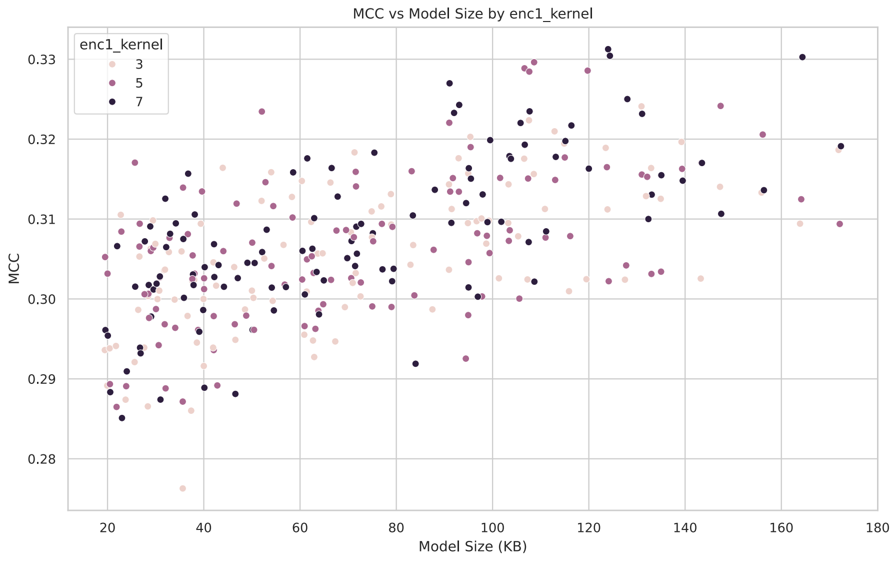


Figure C.28: Relationship between model size and MCC for different `enc1_kernel` values.

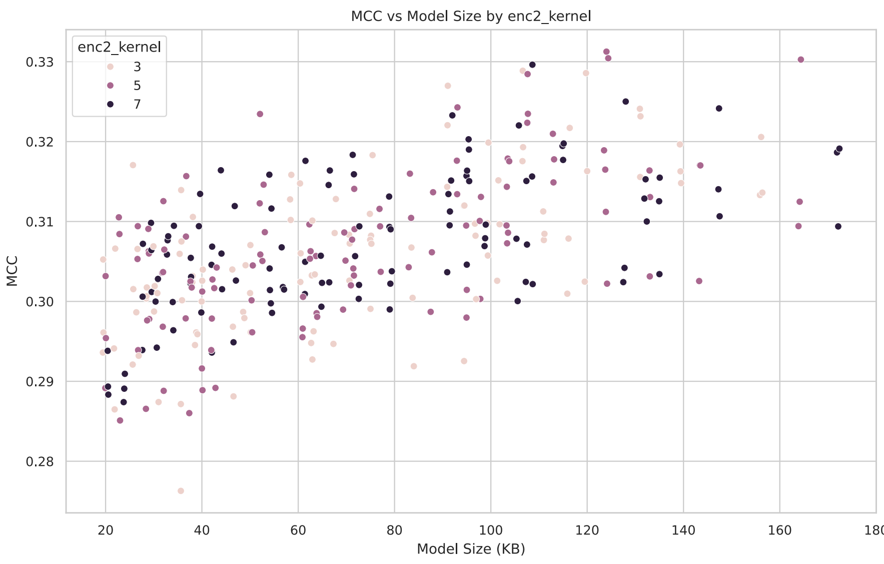
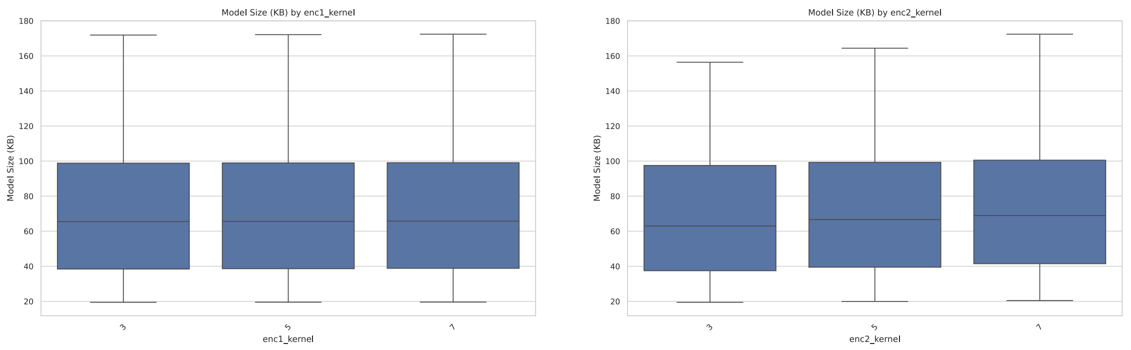


Figure C.29: Relationship between model size and MCC for different `enc2_kernel` values.



(a) Distribution of model sizes for different `enc1_kernel` values.

(b) Distribution of model sizes for different `enc2_kernel` values.

Figure C.30: Model size distributions for varying `enc1_kernel` and `enc2_kernel` values in the Dilated CNN encoder.

Result Tables D

Table D.1: Summary of the comparative experiment results with MCC values across all 96 evaluated configurations for each sensor location.

Location	**Mean**	**Median**	**Std**	**Max**
Tragus	0.0646	0.0621	0.0171	0.1202
Ear Canal (Shallow)	0.1612	0.1690	0.0284	0.2079
Concha	0.2567	0.2598	0.0241	0.3167
Ear Canal (Deep)	0.3041	0.3078	0.0239	0.3609

Table D.2: Summary of MCC results and model sizes across dynamic encoder-decoder CNN for different configuration parameters.

	MCC				**Model size (kB)**	
Configuration	**Mean**	**Median**	**Min**	**Max**	**Mean**	**Range**
`pool_every`						
1	0.127	0.126	0.022	0.207	67.5	1.9–857.1
2	0.233	0.239	0.158	0.276	67.5	1.9–857.1
4	0.223	0.237	0.090	0.282	67.5	1.9–857.1
8	0.220	0.230	0.089	0.284	67.5	1.9–857.1
Pooling type						
Average (`avg`)	0.202	0.225	0.046	0.275	67.5	1.9–857.1
L_p-norm (`lp_2`)	0.200	0.224	0.022	0.284	67.5	1.9–857.1
Max (`max`)	0.200	0.224	0.042	0.280	67.5	1.9–857.1
`base_channels`						
8	0.230	0.236	0.100	0.284	27.43	1.9–216.1
16	0.238	0.244	0.110	0.276	107.61	6.8–857.1
32	0.240	0.250	0.108	0.271	96.52	25.6–304.9
64	0.243	0.251	0.112	0.292	380.47	99.3–1209.8
`double_every`						
1	0.238	0.245	0.110	0.284	287.37	8.4–857.1
2	0.239	0.249	0.102	0.292	194.54	2.8–1209.8
4	0.237	0.245	0.101	0.272	76.61	1.9–406.3
8	0.236	0.244	0.100	0.273	65.75	1.9–325.3
`kernel_size`						
3	0.224	0.233	0.094	0.264	12.78	1.9–47.3
5	0.244	0.245	0.208	0.274	13.57	3.0–77.4
7	0.251	0.252	0.221	0.275	18.69	4.0–107.6
9	0.255	0.256	0.222	0.283	23.82	5.1–137.7
`dilation`						
1	0.232	0.241	0.094	0.275	13.74	1.9–137.7
2	0.249	0.251	0.182	0.283	13.74	1.9–137.7
4	0.250	0.250	0.223	0.279	13.74	1.9–137.7
8	0.242	0.243	0.214	0.274	13.74	1.9–137.7

Table D.3: Summary of MCC results and model sizes across dilated CNN models for different configuration parameters.

	MCC				Model size (kB)	
Configuration	**Mean**	**Median**	**Min**	**Max**	**Mean**	**Range**
`num_dilated_layers`						
3	0.294	0.292	0.247	0.339	208.99	11.6–786.0
5	0.315	0.307	0.276	0.373	380.40	18.0–1429.0
7	0.308	0.307	0.276	0.364	452.16	24.4–1998.3
`bottleneck_channels`						
16	0.288	0.288	0.247	0.307	31.36	11.6–60.8
32	0.299	0.302	0.274	0.322	88.46	33.2–168.1
64	0.310	0.310	0.274	0.353	298.67	112.3–562.8
128	0.327	0.328	0.283	0.373	1017.33	414.6–1998.3
`dilated_kernel`						
3	0.288	0.290	0.269	0.307	42.38	18.0–69.0
5	0.295	0.294	0.279	0.317	64.63	27.0–105.0
7	0.297	0.298	0.275	0.323	86.88	36.0–141.0
9	0.301	0.300	0.279	0.321	109.13	45.0–177.0
`base_dilation`						
1	0.290	0.288	0.270	0.320	75.75	18.0–177.0
2	0.301	0.300	0.282	0.323	75.75	18.0–177.0
3	0.296	0.296	0.269	0.317	75.75	18.0–177.0
4	0.294	0.296	0.273	0.316	75.75	18.0–177.0
`enc1_channels`						
8	0.307	0.306	0.276	0.330	63.81	19.4–135.1
16	0.306	0.306	0.285	0.324	70.01	21.8–147.5
32	0.307	0.307	0.286	0.331	82.43	26.4–172.4
`enc2_channels`						
8	0.303	0.303	0.285	0.322	55.06	19.4–105.8
16	0.307	0.306	0.276	0.331	67.83	25.6–128.0
32	0.310	0.310	0.288	0.330	93.37	38.0–172.4
`enc1_kernel`						
3	0.306	0.306	0.276	0.324	71.94	19.4–171.9
5	0.306	0.306	0.286	0.330	72.08	19.5–172.1
7	0.308	0.307	0.285	0.331	72.23	19.6–172.4
`enc2_kernel`						
3	0.306	0.306	0.276	0.329	69.36	19.4–156.4
5	0.307	0.306	0.285	0.331	72.08	19.9–164.4
7	0.307	0.307	0.287	0.330	74.81	20.4–172.4