International Journal of
Network Dynamics and Intelligence (IJNDI)

Article

Modeling and Evaluating an Intelligent Health Monitoring System for Detecting Atrial Fibrillation

Petter Nordin [1], Hossein Fotouhi [2,*], Miguel Leon [2], Oana Cramariuc [3], Tiberiu Seceleanu [2], and Maryam Vahabi [2]

[1] Software Developer TRATON Group, 15139 Södertälje, Sweden
[2] Department of Computer Science and Engineering, Mälardalen University (MDU), 72220 Västerås, Sweden
[3] IT Centre for Science and Technology, 060042 Bucharest, Romania
* Correspondence: hossein.fotouhi@mdu.se

Received: 15 September 2025; Revised: 27 February 2026; Accepted: 7 March 2026; Published: 16 March 2026

Abstract: Atrial Fibrillation (AFib) is a common cardiac arrhythmia whose global prevalence has risen in recent years. If left untreated, AFib can lead to severe complications such as stroke and heart failure. Because AFib may occur without symptoms, continuous monitoring is critical for timely detection. This paper presents a low-cost Intelligent Health Monitoring System (IHMS) that uses one-dimensional Convolutional Neural Networks (1D-CNNs) to detect AFib from Electrocardiogram (ECG) signals. The study evaluates the feasibility of deploying 1D-CNN models on resource-constrained edge devices and compares edge- and cloud-based computing architectures with respect to inference efficiency. Three 1D-CNN models of increasing complexity are designed, trained, and tested on datasets containing AFib and Normal Sinus Rhythm (NSR) segments. Two experiments are conducted to assess end-to-end delay and prediction time under a controlled experimental setup. The results demonstrate the potential for on-device AFib detection in constrained environments and provide practical insights into selecting suitable architectures for embedded deployment.

Keywords: edge computing (EC); health monitoring; atrial fibrilation (AFib); machine learning (ML)

1. Introduction

Atrial fibrillation (AFib) [1] is the most common clinically significant cardiac arrhythmia and its global prevalence continues to rise. In 2010, an estimated 35 million individuals were diagnosed worldwide [2], with projections suggesting 6–12 million cases in the United States by 2050 and 17.9 million cases in Europe by 2060 [3]. AFib is characterized by rapid, irregular atrial activity caused by disorganized electrical impulses in the atria, resulting in loss of coordinated contraction. This disruption reduces cardiac pumping efficiency and significantly increases the risk of thromboembolism, stroke, heart failure, and other cardiovascular complications.

Early detection is essential, as AFib may present with noticeable symptoms or remain entirely asymptomatic. Undetected or untreated AFib can significantly impair quality of life and increase mortality risk [4]. The gold standard for diagnosis is the Electrocardiogram (ECG), where clinicians identify deviations from Normal Sinus Rhythm (NSR). NSR is characterized by a repeating sequence of P-wave, QRS complex, and T-wave with consistent amplitude and timing. AFib is typically indicated by the absence of P-waves, irregular atrial activity, and an irregular RR interval. The RR interval is defined as the duration between two consecutive R peaks of the QRS complex, which is regular in the NSR—see Figure 1.

Traditionally, AFib diagnosis has relied on hospital-grade ECG equipment and expert interpretation. While clinically reliable, this approach is constrained by the limited availability of trained personnel and the intermittent nature of clinical recordings. Short-term ECG snapshots taken in clinical settings may fail to capture paroxysmal AFib, as the arrhythmia might not manifest during the brief recording window [4–8]. Consequently, there is increasing interest in automated AFib detection systems capable of continuous monitoring through affordable, portable, and wearable technologies.

A broad range of AFib detection methods has been proposed. Early approaches focused on handcrafted features extracted from ECG signals, including time-domain descriptors (e.g., RR-interval variability), frequency-domain characteristics, and statistical measures of signal regularity. More recently, end-to-end deep learning models—particularly one-dimensional Convolutional Neural Networks (1D-CNNs)—have shown a strong capability in learning discriminative representations directly from raw ECG signals. However, many of these methods rely on cloud-based processing or high-performance computing hardware, limiting their suitability for real-time deployment in low-cost, resource-constrained monitoring systems.

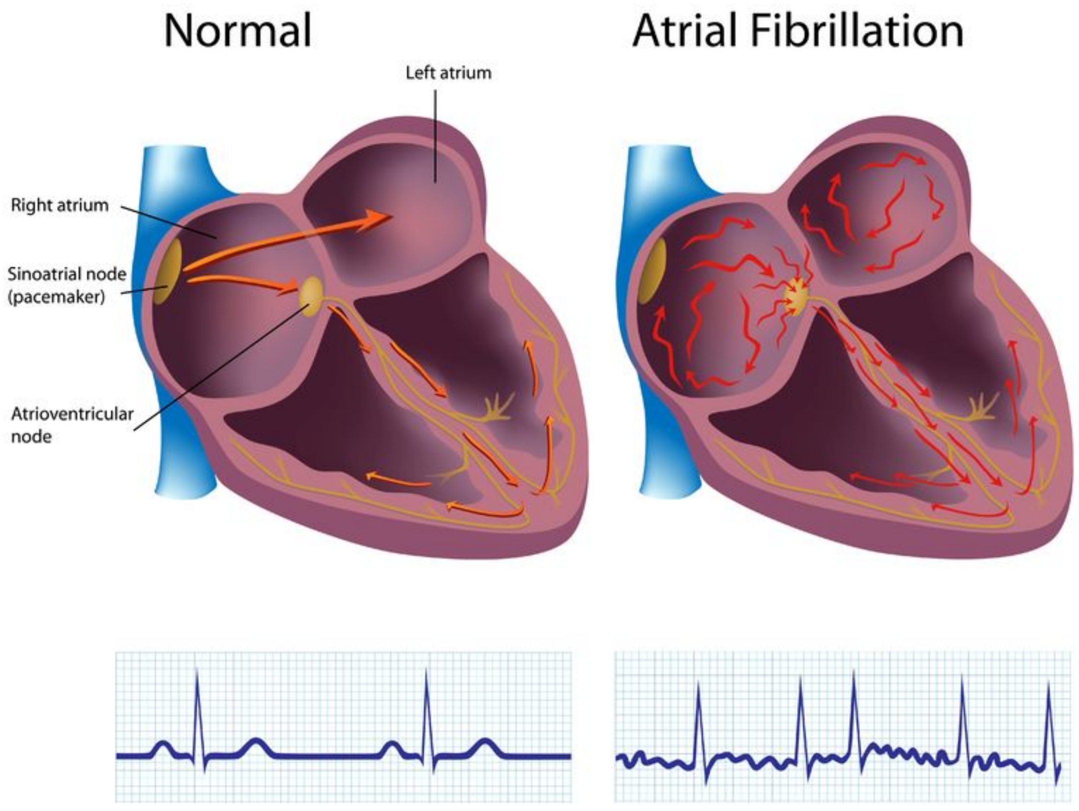

Figure 1. NSR (**left**) vs. AFib (**right**).

Edge computing has emerged as a promising paradigm for enabling continuous, on-device AFib detection with reduced latency, lower bandwidth usage, and improved data privacy. Yet, only a limited number of studies have assessed the feasibility of deploying deep learning models for AFib detection on resource-constrained devices such as the Raspberry Pi.

The main contributions of this work are: (i) the development of a low-cost Intelligent Health Monitoring System (IHMS) for AFib detection; (ii) a comparative evaluation of edge and cloud computing architectures in terms of end-to-end delay and prediction latency; (iii) identification of the most suitable 1D-CNN model for deployment on a resource-constrained edge device; and (iv) design of a low-complexity 1D-CNN architecture optimized for real-time AFib detection.

To support these contributions, the remainder of the paper is organized as follows. Section 2 reviews related work on AFib detection. Section 3 introduces the overall system architecture, detailing the ECG input data, preprocessing steps, CNN model design, and training procedure. Section 4 outlines the performance evaluation strategy, including both model performance metrics and the experimental setup for measuring system-level latency on cloud and edge platforms. Section 5 presents and discusses the results, covering classification outcomes, latency benchmarks, practical deployment implications, physician feedback, and key limitations. Finally, Section 6 offers concluding remarks and outlines future directions.

2. Related Work

Atrial fibrillation (AFib) detection has been widely studied using both traditional signal processing techniques and deep learning models. Time-domain and statistical approaches such as RR-interval analysis have proven effective in identifying irregular cardiac rhythms associated with AFib [9]. The proliferation of consumer-grade wearable devices has further enabled large-scale rhythm monitoring using photoplethysmography (PPG) signals in conjunction with deep learning models [10]. In parallel, multimodal frameworks have been developed to detect AFib from wearable ECG data [11]. Furthermore, Convolutional Neural Networks (CNNs) and other deep learning methods have demonstrated high classification accuracy across a range of ECG interpretation tasks [12,13].

Several 1D-CNN-based methods have been developed for AFib and arrhythmia detection. Pourbabaee et al. [14]

proposed a five-layer 1D-CNN combined with hybrid classifiers (kNN and SVM) for paroxysmal AFib detection. Andersen et al. [15] introduced a hybrid CNN–LSTM model that achieved strong AFib classification performance using RR intervals. EdgeCNN [16] presented a hybrid edge–cloud architecture for ECG analysis, combining edge preprocessing with cloud-based deep learning inference. In the domain of IoT, machine intelligence has been applied to ECG analysis using cloud-assisted solutions [17], and lightweight Artificial Neural Network (ANN) approaches have been developed for low-cost AFib detection [18]. Real-time ECG beat classification on edge devices has been explored in [19], while AfibNet [20] introduced a multi-branch CNN optimized for AFib detection. Xia et al. [21] used deep 2D CNNs on time–frequency representations to classify AFib and other arrhythmias.

Taken together, these representative AFib and arrhythmia detection systems illustrate the trade-offs between network capacity, classification performance, and deployment assumptions. The five-layer 1D-CNN in [14] achieves up to 91% correct classification in combination with kNN, but it is evaluated offline. Andersen et al. [15] reported 97.10% accuracy on AFDB using a deeper CNN–LSTM hybrid operating on RR-interval sequences, yet do not examine latency or feasibility on constrained hardware. EdgeCNN [16] includes five convolutional blocks and reports an F1 score of approximately 78% for AFib-related classes, but still depends on cloud offloading and does not quantify model size. AfibNet [20] and the deep 2D CNN [21] both leverage higher-capacity architectures to improve detection performance, but are evaluated entirely on desktop or cloud platforms without systematic assessment of on-device computational cost. In contrast, the present work focuses on lightweight 1D-CNN architectures and quantifies their prediction latency, end-to-end delay, and deployability on a resource-constrained Raspberry Pi 3B+.

Beyond classical AFib-specific models, recent works have expanded deep-learning-based ECG classification through improved architectures and training methods. Khan et al. [22] developed a deep residual 1D-CNN with SMOTE balancing for multi-class arrhythmia detection, achieving 98.63% accuracy. Ahmed et al. [23] demonstrated strong performance using compact 1D-CNN models evaluated on the MIT-BIH database. Similarly, Mallikarjunamallu et al. [24] introduced a three-step denoising and classification pipeline using Gaussian noise injection, notch filtering, and 1D-CNN classification to improve classification robustness. Although accurate, none of these studies examine inference latency or real-time deployment on low-power edge devices.

Transforming or restructuring ECG signals for more expressive spatial representations has been explored in several recent works. Ref. [25] proposed a reconstruction-based CNN using multiscale representation learning and squeeze-and-excitation mechanisms for AFib detection. Extending this direction, Gao et al. [26] introduced an efficient 1D-to-2D mapping method enabling the use of 2D CNNs for AFib detection. Though performant, both approaches rely on computationally intensive preprocessing and 2D convolution operations that are less suitable for constrained edge platforms.

Privacy-preserving ECG classification has also been investigated using distributed learning paradigms. Elmir et al. [27] employed federated learning with Gramian Angular Fields to train ECG classifiers across heterogeneous IoT devices without exchanging raw data. While effective for data confidentiality, the work does not analyze inference timing or deployability on limited-resource hardware.

A separate line of research focuses on hardware-accelerated ECG classification. Ahmed et al. [28] deployed a quantized-pruned 1D-CNN on an FPGA-enabled System-on-Chip (SoC), achieving lower power consumption and improved throughput. Such FPGA-based solutions, however, require specialized hardware not representative of the low-cost general-purpose edge devices targeted in this work.

Advanced hybrid architectures have also emerged. Guhdar et al. [29] integrated temporal attention mechanisms into a 1D-CNN, improving discriminative capability but at higher computational cost and without evaluation on embedded devices.

Overall, while recent research demonstrates strong ECG classification performance, most approaches depend on computationally intensive architectures, specialized hardware, or cloud-based inference. Only limited studies examine real-time inference latency, end-to-end delay, or practical feasibility on low-cost edge platforms. This study addresses this gap by performing a structured evaluation of three lightweight 1D-CNN model variants with different depths, and measuring their classification performance, prediction time, and end-to-end delay on a Raspberry Pi 3B+.

3. AFib Detection: System Architecture and Framework

In this work, we develop a low-cost IHMS approach for detecting AFib from ECG signals using a deep learning model based on a 1D-CNN architecture. We begin by defining and executing a set of subtasks, including signal preprocessing, model training and evaluation, and deployment on cloud and edge platforms to assess latency. Preprocessing involves selecting and applying appropriate filters to suppress Baseline Wander (BLW), Powerline Interference (PLI), and Electromyogram (EMG) noise, followed by segmentation of the ECG into fixed-length intervals suitable for model input. The training phase involves developing three 1D-CNN models of varying complexity to classify AFib versus NSR. These models are then deployed on both edge and cloud infrastructures to compare prediction times and end-to-end delays. In parallel, we investigate methods for ECG feature extraction relevant to AFib detection, with a focus on QRS and P-wave localization.

One advantage of using deep learning models such as CNNs is that they can operate directly on raw ECG signals, minimizing the need for extensive feature engineering. However, basic preprocessing—particularly noise removal—remains essential to ensure signal quality. In this work, we rely on preprocessed raw ECG signals rather than derived features (e.g., RR intervals) to evaluate both the strengths and limitations of this approach. We also explore whether incorporating auxiliary features such as RR-interval variability could enrich the input representation.

3.1. Input Data: ECG Signal

The ECG data used in this study are sourced from the Long-Term AF Database (LTAFDB) [30,31], which contains 84 extended ECG recordings from patients with paroxysmal or sustained atrial fibrillation. Each record includes two-lead ECG signals sampled at 128 Hz. The dataset follows the PhysioNet standard format, where each record comprises three files: a signal file (.dat), a header file (.hea), and an annotation file (.atr) (PhysioNet FAQ. Available online: https://archive.physionet.org/faq.shtml (accessed on 10 March 2026)). These files enable the reconstruction of both digital and physical ECG signals and provide metadata for rhythm and beat annotations. A comprehensive reference of beat and rhythm types, including annotation codes and descriptions, is available through the official PhysioNet annotation tables (Reference annotations: https://archive.physionet.org/physiobank/database/ltafdb/tables.shtml (accessed on 10 March 2026); Annotation descriptions: https://archive.physionet.org/physiobank/annotations.shtml (accessed on 10 March 2026)).

3.2. Data Pre-Processing

The ECG signal pre-processing consists of two main steps: (i) noise removal to eliminate baseline wander (BLW), powerline interference (PLI), and electromyogram (EMG) artifacts, and (ii) signal segmentation. A subset of 25 records is selected from the LTAFDB based on the presence of both AFib and NSR episodes. The selected record IDs are: 00, 01, 05, 07, 08, 10, 13, 19, 23, 24, 35, 39, 47, 53, 55, 56, 58, 72, 101, 103, 112, 114, 115, 117, and 122. For reading and processing these records, we use the WFDB Python package [32], which provides tools for handling PhysioNet-formatted signal and annotation files.

For each selected record, we extract rhythm and beat information from the annotation file (.atr). We begin by identifying the minimum and maximum sample numbers in the signal. Next, we locate the indices of rhythm change annotations and determine their corresponding sample numbers and beat types. QRS complexes are identified by filtering out rhythm symbols that do not include the marker "(", and their sample indices are likewise extracted. All rhythm change and QRS indices are then adjusted by subtracting the first sample index, ensuring consistent alignment across records. This processed information is stored for efficient access to the raw signal data in the .dat files and for precise segmentation of NSR and AFib episodes. Notably, rhythm annotations are extracted from the aux_note variable provided by the rdann function in the WFDB Python module.

We continue by reading the two-lead digital ECG signal and the .dat file, from the minimum to the maximum sample number. Using the rhythm change annotations (AFIB and (N, we identify and extract the durations of AFib and NSR episodes. These signal rhythm episodes are segmented into batches of 10 s, corresponding to 1280 data points per segment at the 128 Hz sampling rate. These AFib and NSR segments are saved into separate files for each record.

To further prepare the signal data for CNN model training, we first normalized each ECG trace to the [0,1] range to preserve morphology while reducing computational load. We then applied a first-order Butterworth bandpass filter with cut-off frequencies of 5 Hz and 15 Hz (for a 128 Hz sampling rate, these correspond to normalized cut-offs $5/128$ and $15/128$). This relatively narrow 5–15 Hz band was chosen to emphasize the QRS complex frequency content while attenuating noise outside this range. This filtering strategy prioritizes rhythm irregularity—through reliable R-peak localization—over detailed ECG morphology, which is less critical for distinguishing AFib from NSR. In particular, frequencies below 5 Hz (baseline wander and very slow components) and above 15 Hz (muscle artifact and high-frequency noise) are greatly suppressed, which is a common practice in QRS-detection literature (as in [33]). The 15 Hz low-pass cutoff also implicitly removes powerline interference (50/60 Hz), without the need for a dedicated notch filter. We acknowledge that using a 5–15 Hz bandpass entails trade-offs. Such a narrow bandwidth attenuates certain physiological components of the ECG; for example, the low-frequency portions of P-waves and T-waves, as well as very high-frequency details of the QRS complex, may be diminished. Our band selection was guided by traditional QRS-detection methods (rather than a new power spectral analysis of the present dataset), with the intent to retain the dominant QRS energy for heartbeat detection while suppressing baseline drift and broadband noise. For AFib versus NSR classification, this compromise is acceptable because the task relies primarily on R-peak timing and rhythm irregularity rather than on fine-grained wave morphology. Empirical inspection of sample segments indicated that even this simple first-order filter effectively removed most noise while preserving the key discriminative features needed for AFib detection. For diagnostic application, a broader bandwidth (e.g., standard 0.5–40 Hz) or a higher-order filter may be preferred to fully preserve wave details.

It is also worth noting that more advanced noise-suppression techniques, such as discrete wavelet denoising, adaptive filtering, or empirical mode decomposition, can provide stronger attenuation of high-frequency EMG artifacts. However, these approaches typically involve substantially higher computational cost and memory usage, making them less suitable for real-time execution on a Raspberry Pi 3B+. As embedded processors continue to improve in efficiency, such methods may become viable for on-device AFib detection, but they were beyond the practical scope of the present feasibility-oriented study. The filtered, normalized AFib and NSR segments are used as input for model training.

3.3. Model Architecture

Three 1D-CNN models were designed using the Keras functional API [34], with increasing depth and identical internal structure. Each model is composed of a sequence of convolutional blocks (ConvBlocks), followed by fully connected layers and a binary classification output. As illustrated in Figure 2, each ConvBlock contains two consecutive 1D convolution layers with five filters (kernel size 5), a batch normalization layer, a LeakyReLU activation, and a max-pooling operation (pool size 2, stride 1). The three models consist of 4–6 such blocks, respectively.

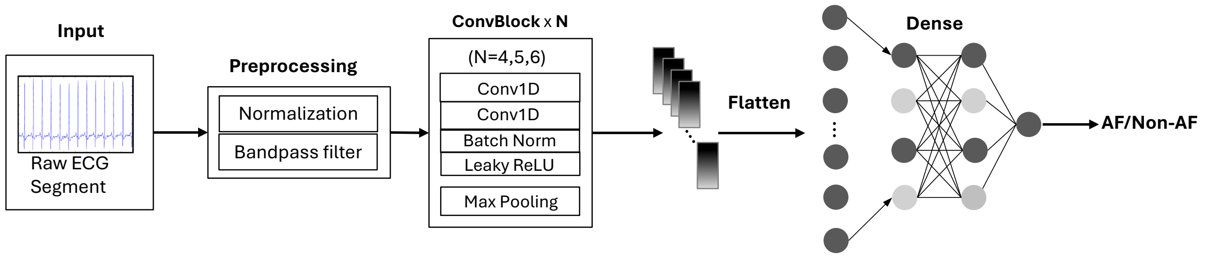

Figure 2. Overview of the proposed 1D-CNN architecture for AFib detection on edge devices.

The choice of architecture depth was guided by a trade-off between temporal feature extraction capacity and the deployment constraints of a Raspberry Pi 3B+. In 1D-CNN-based ECG classification, deeper CNNs allow for larger receptive fields and more abstract representations. However, increasing model complexity can exceed the memory and latency limits of embedded hardware. Pilot experiments indicated that models with fewer than four blocks consistently underfit the data, while deeper variants imposed excessive inference delays. The selected models thus represent a practical balance between classification performance and real-time deployability on constrained hardware.

All models share the same fully connected head: two dense layers with four neurons (LeakyReLU activation) and a final output layer for binary classification. Dropout (rate 0.5) and max-norm regularization (constraint 4) are applied to both convolutional and dense layers to reduce overfitting. The input shape is $4 \times 1280 \times 1$ (batch size × signal length × features), and hyperparameters were tuned via coarse grid search. Architectures with larger widths or depths were excluded based on memory and latency profiling on the edge device.

3.4. Model Training

Model training was performed using TensorFlow 2.8 on a desktop workstation with an Nvidia GeForce RTX 2060 GPU, CUDA 11.2, and cuDNN 8.1. A 5-fold cross-validation was used, with 15 records for training/validation and 10 records for testing. AFib and NSR labels were one-hot encoded, and categorical cross-entropy was used as the loss function.

To address class imbalance, Synthetic Minority Over-sampling Technique (SMOTE) [35] was applied to the fixed-length ECG segments within each fold. Although SMOTE is not specifically designed for time-series data, it was used here under the assumption that rhythm irregularity—rather than detailed waveform morphology—is the key discriminative feature for AFib classification. Visual inspection of synthetic samples revealed no artifacts, and classification performance remained stable.

The models were trained for 30 epochs using the ADAM optimizer (learning rate 0.01), with early stopping applied (patience 10 epochs, based on validation loss). On average, training converged within 10–15 epochs. A batch size of 4 was used, constrained by GPU memory limits. Session clearing and memory deallocation (via Keras backend and Python garbage collection) were necessary to avoid memory overflow during extended training cycles. Training each fold required approximately 10–15 h.

4. Performance Evaluation and Deployment Analysis

This section presents a twofold evaluation of the proposed AFib detection framework. First, we analyze the classification performance of three 1D-CNN models under three different test distributions. Then, we detail the deployment of these models on a resource-constrained testbed and describe the procedures implemented to support the measurement of system-level latency,including both end-to-end delay and prediction time, across edge and cloud platforms. This two-pronged analysis provides a foundation for understanding the trade-offs between detection accuracy, model complexity, and real-time feasibility in practical deployment scenarios.

4.1. Model Performance Evaluation

We evaluate the performance of the three 1D-CNN models in classifying AFib and NSR using a 10-record test set. To assess robustness under varying class distributions, we conduct experiments across three test configurations: oversampled, undersampled, and imbalanced.

In the oversampling setting, we generate 10 iterations of balanced test sets using SMOTE [35], an oversampling technique that synthesizes samples from the minority class to equalize class distribution. In each iteration, the final model from every cross-validation fold is evaluated using the following metrics: Area Under the ROC Curve (AUC), accuracy, sensitivity, specificity, F1-score, and Matthews Correlation Coefficient (MCC). Final results are reported as the average performance across all folds. The undersampling experiment follows the same evaluation protocol, except that class balancing is achieved using RandomUnderSampler [36], which removes samples from the majority class to achieve parity. Finally, in the imbalanced setting, we evaluate model performance on the original, unbalanced test distribution without any resampling. Metrics are again computed for each fold and then averaged.

This evaluation protocol is applied to all three CNN models to enable a fair comparison of performance under varying class distributions.

4.2. Experimental Setup for System-Level Latency

To evaluate the feasibility of deploying the proposed AFib detection framework in real-world scenarios, we conducted experiments to measure two critical timing metrics: end-to-end delay and prediction time. These experiments were designed to assess performance in both edge and cloud computing settings and to investigate the trade-offs between model complexity, inference speed, and detection accuracy.

The experimental setup consists of three entities connected to the same local network: a laptop acting as an independent data source, a Raspberry Pi 3B+ serving as the edge computing platform, and a cloud platform comprising Google Drive and Google Colaboratory (https://colab.google/ (accessed on 10 March 2026)). The end-to-end delay experiment measures the time required to transmit ECG segment files from the laptop to the processing platform (either edge or cloud). The prediction time experiment measures how long each deployed 1D-CNN model takes to infer a prediction once it receives the ECG input.

These experiments were motivated by two practical goals. First, we aimed to assess whether computationally constrained devices such as the Raspberry Pi 3B+ can support real-time inference for AFib detection. Second, we sought to evaluate how different CNN model complexities affect the balance between detection performance and responsiveness. The combined evaluation of end-to-end delay and prediction time provides insight into the system's suitability for latency-sensitive health monitoring applications.

To prepare the test inputs, we generated 10-second ECG segments from pre-processed signals of patient 122 in the LONG TERM AF database [30,31]. This patient was selected because their data appears in the test set across all three models. Eight segment group sizes were created—4, 8, 12, 16, 24, 32, 64, and 128 segments per file—for both AFib and NSR episodes, resulting in 16 distinct ECG file types used during evaluation.

For the end-to-end delay experiments, files were transmitted from the laptop to either the Raspberry Pi or the Google Drive platform. Each rhythm type (AFib and NSR) was represented by up to 100 files per group size, with fewer files for larger segment sets. During each file transfer, two timestamps were recorded on the laptop: one at the time of transmission and another at the moment the file's successful receipt was confirmed. The delay was computed as the difference between these timestamps. This design was necessary for two reasons. First, the clocks of the laptop and the Raspberry Pi were not synchronized, making cross-device time measurements unreliable. Second, Google Drive and its API support only indirect confirmation of receipt—requiring metadata queries based on file ID to obtain the upload timestamp.

4.2.1. Cloud-Based End-to-End Delay Setup

To assess cloud-based communication latency, the experiment is designed to measure end-to-end delay between an independent data source (laptop) and the cloud computing platform (Google Drive, integrated with Google Colaboratory). This setup enables asynchronous file transfer followed by server-side processing. Communication is handled using the Google Drive API v3, following standard authentication and file management procedures as described in [37].

A custom Python class was developed to manage this process. The first method handles authentication to Google Drive using port 8090 and retrieves the necessary service object. The second method uploads ECG segment files, attaching metadata including the filename, parent folder ID, and timestamp (createdTime) in ISO 8601 format with UTC time zone. Uploads are handled using the MediaFileUpload object, configured for resumable transfers. The third method retrieves the createdTime of the most recently uploaded file by querying the Drive API with the filename, enabling timestamp-based delay measurement.

For each file upload, two timestamps are recorded: one at the client side (prior to upload) and another extracted from the uploaded file's metadata in Google Drive. The end-to-end delay is calculated as the difference between these two timestamps. This process is repeated for multiple files of varying sizes—ranging from 4 to 128 ten-second ECG segments—to evaluate how data volume impacts transfer latency. Each segment file is encrypted using Fernet symmetric encryption before upload.

To ensure consistency, the experiment is repeated across different segment batch sizes, with 100 AFib and 100 NSR samples per type where feasible. After all uploads, the minimum and average delay values are computed. This timestamp-based approach is necessary due to the lack of clock synchronization between the sender device and the cloud platform, and due to limitations of the Google Drive API in providing more precise delivery acknowledgments.

4.2.2. Edge-Based End-to-End Delay Setup

The end-to-end delay experiment on the edge platform, implemented on a Raspberry Pi 3B+, utilizes Python socket programming to enable file transmission between a sender (a laptop) and a receiver (the edge device). For each experiment, a specific ECG segment file type is selected, and 100 files are prepared per cardiac rhythm—AFib and NSR. To facilitate the communication and data transfer, a dedicated Python class named `IoTLayer` is developed. This class establishes a TCP socket connection to the Raspberry Pi using its IP address on port 5001. A complementary `Buffer` class is used to manage data transmission efficiently, employing a buffer size of 5 MB.

The `Buffer` class is responsible for handling both byte-level and UTF-8 string-based communication. It reads incoming data from the socket in buffered chunks and assembles complete messages, while also supporting reliable data transmission through explicit send operations. To support UTF-8 encoded control messages, the buffer includes logic to identify null-terminated strings and decode them appropriately. For sending, strings are encoded in UTF-8 and terminated with a null character to mark message boundaries.

On the edge device, an `Edge` class is deployed to receive and reconstruct incoming ECG files. It establishes a listening socket on the specified port and accepts connections from the independent data source. Once a connection is established, the system reads the incoming filename and file size, creates a local file on disk, and receives the file contents in buffered chunks. The received bytes are written to disk until the entire file is received. A confirmation message is then sent back to the sender indicating successful receipt.

Prior to transmission, each ECG segment file is encrypted using symmetric encryption (Fernet) for security. A timestamp is recorded immediately before the file is transmitted from the laptop. Upon receipt of the confirmation message from the Raspberry Pi, a second timestamp is generated. The difference between these two timestamps is used to calculate the end-to-end delay for each file transfer. These measurements are collected across all transmissions, and the minimum and average delays are computed for evaluation.

Due to the inability to synchronize clocks between the Windows operating system on the laptop and the Linux-based Raspberry Pi, this timestamping approach—implemented solely on the sender side—was deemed the only reliable method for measuring delay. This setup ensures accurate estimation of transmission latency in the edge computing scenario, forming a critical part of the system-level performance evaluation.

4.2.3. Cloud-Based Prediction Time Setup

Prediction time benchmarking on the cloud platform is carried out using Google Colaboratory, where the three trained 1D-CNN models—CNN4, CNN5, and CNN6—are deployed in both TensorFlow SavedModel and TensorFlow Lite formats. A Python class is implemented to manage the full evaluation pipeline. This class reads input ECG files, each containing four 10-second ECG segments, and feeds them into the deployed models sequentially.

To measure the prediction time, two precise timestamps are recorded: one immediately before invoking the TensorFlow inference function and the other directly afterward. These timestamps are obtained using the `process_time` and `perf_counter` functions from Python's time module to ensure high-resolution time measurement. The difference between the two timestamps yields the total inference time for the batch of four ECG segments.

This procedure is repeated for all three model configurations in both SavedModel and TensorFlow Lite formats to assess the impact of model size and representation on inference efficiency in the cloud environment. The resulting measurements are recorded for later comparison with the corresponding edge device performance.

4.2.4. Edge-Based Prediction Time Setup

On the edge device (Raspberry Pi 3B+), prediction time is measured using a dedicated extension of the Edge class introduced in the end-to-end delay experiments. The class includes two methods: the first loads a 4-segment ECG file, and the second handles inference execution for the three TensorFlow Lite models—CNN4, CNN5, and CNN6.

Prediction time is measured by placing timestamp markers immediately before and after the model inference operation. The same `process_time` and `perf_counter` functions are used to ensure consistency with the cloud setup. Each model processes the same input data as in the cloud experiments, allowing for a direct comparison of execution time across platforms.

This setup allows us to quantify the latency introduced by on-device inference, isolate model execution time from network factors, and assess the suitability of each model for real-time ECG processing on resource-constrained hardware.

5. Results and Discussion

This section presents the evaluation results for both the classification performance of the 1D-CNN models and the system-level latency of the proposed AFib detection framework. The analysis is twofold: (i) we assess the suitability of deploying increasingly complex CNN models on low-cost edge devices for intelligent health monitoring, and (ii) we evaluate end-to-end delay and prediction time across edge and cloud architectures to examine trade-offs in latency and feasibility. We begin by reporting the models' classification performance under varying test distributions, followed by system-level latency results and practical deployment considerations.The section concludes with an analysis of key system limitations.

5.1. Model Performance Results

The models are evaluated on three test set distributions: oversampling (OT), undersampling (UT), and imbalanced (IT). A 5-fold cross-validation is used to train the models, resulting in five instances of each 1D-CNN model. The evaluation metrics include accuracy (ACC), area under the curve (AUC), sensitivity (TPR), specificity (TNR), F1-score (F1), and Matthews correlation coefficient (MCC). F1 and MCC are considered only for evaluation on imbalanced data, as they are not suitable for equally distributed datasets (UT and OT). Conversely, ACC is considered only for the equally distributed data.

The performance of the models across three experiments, denoted CNN4, CNN5, and CNN6, is presented in Tables 1–3. Tables 4–6 provide a performance comparison of the models for each experiment. In addition to evaluating the models, these tables are used to identify the most suitable reference model candidate for deployment in the prediction time experiment. The CNN model that demonstrates the highest overall performance across most metrics in the three experiments is selected as the ideal candidate.

5.1.1. CNN4 Evaluation

As shown in Table 1, no single model outperforms others across all metrics. However, the 5th CNN4 model demonstrates superior performance across the majority of metrics and is therefore selected as the ideal candidate for the prediction time experiment. Alternatively, rather than selecting the model with the highest scores in individual metrics, one could opt for a more stable model—one that performs consistently well across all metrics without necessarily excelling in any single one. The 4th CNN4 model exhibits these characteristics.

5.1.2. CNN5 Evaluation

Based on the results presented in Table 2, the first instance of CNN5 demonstrates the highest overall performance across evaluation metrics, particularly in detecting AFib episodes. The differences in NSR classification accuracy across models are marginal. Consequently, CNN5–Fold 1 is selected for use in the prediction time experiments. While the third instance exhibits more uniform performance across metrics—suggesting greater stability—this consistency does not outweigh the superior classification performance of the first model, especially given that stability does not necessarily translate into improved or faster inference.

Table 1. Performance of CNN4. Bold values indicate the best performance across the compared models.

Experiment	ACC	AUC	TPR	TNR	F1	MCC
OT						
CNN4:1	**0.7372**	0.8103	0.9130	0.5614	**0.7765**	**0.5068**
CNN4:2	0.6774	0.7821	0.4312	**0.9236**	0.5720	0.4077
CNN4:3	0.7068	0.7991	0.5602	0.8533	0.6564	0.4326
CNN4:4	0.7271	0.7624	0.8032	0.6511	0.7464	0.4597
CNN4:5	0.7060	**0.8407**	**0.9778**	0.4341	0.7688	0.4909
UT						
CNN4:1	0.8188	0.9032	0.9136	0.7240	0.8345	0.6494
CNN4:2	0.7119	0.8321	0.4306	**0.9931**	0.5991	0.5126
CNN4:3	0.7713	0.8779	0.5610	0.9816	0.7104	0.5981
CNN4:4	**0.8362**	0.8987	0.8025	0.8698	0.8304	0.6739
CNN4:5	0.8351	**0.9411**	**0.9779**	0.6924	**0.8557**	**0.6994**
IT						
CNN4:1	0.8766	0.9404	0.9130	0.7240	0.9227	0.6171
CNN4:2	0.5394	0.6062	0.4312	**0.9931**	0.6019	0.3510
CNN4:3	0.6414	0.7289	0.5602	0.9816	0.7161	0.4290
CNN4:4	0.8160	0.8654	0.8032	0.8698	0.8758	0.5660
CNN4:5	**0.9228**	**0.9810**	**0.9778**	0.6924	**0.9534**	**0.7377**
Avg						
Avg OT	0.7109	0.7989	0.7370	0.6847	0.7040	0.4595
Avg UT	**0.7946**	**0.8906**	**0.7371**	0.8521	0.7660	**0.6266**
Avg IT	0.7592	0.8244	**0.7371**	**0.8522**	**0.8140**	0.5402

Table 2. Performance of CNN5. Bold values indicate the best performance across the compared models.

Experiment	ACC	AUC	TPR	TNR	F1	MCC
OT						
CNN5:1	0.8302	**0.9195**	**0.9334**	0.7271	0.8461	0.6751
CNN5:2	0.8182	0.8621	0.7704	0.8659	0.8090	0.6393
CNN5:3	0.7849	0.8464	0.8303	0.7395	0.7942	0.5722
CNN5:4	0.8302	0.8839	0.7526	**0.9077**	0.8159	0.6685
CNN5:5	**0.8463**	0.9115	0.9295	0.7631	**0.8581**	**0.7025**
UT						
CNN5:1	**0.8696**	**0.9391**	**0.9334**	0.8057	**0.8774**	**0.7453**
CNN5:2	0.8130	0.8595	0.7699	0.8561	0.8046	0.6284
CNN5:3	0.8392	0.9038	0.8310	0.8474	0.8379	0.6785
CNN5:4	0.8594	0.9057	0.7527	**0.9662**	0.8426	0.7359
CNN5:5	0.8629	0.9281	0.9297	0.7961	0.8715	0.7324
IT						
CNN5:1	**0.8863**	**0.9508**	**0.9334**	0.8057	**0.9120**	**0.7533**
CNN5:2	0.8020	0.8397	0.7704	0.8561	0.8308	0.6064
CNN5:3	0.8366	0.9046	0.8303	0.8474	0.8651	0.6626
CNN5:4	0.8314	0.8784	0.7526	**0.9662**	0.8492	0.6938
CNN5:5	0.8803	0.9423	0.9295	0.7961	0.9074	0.7401
Avg						
Avg OT	0.8220	0.8847	0.8432	0.8007	0.8247	0.6515
Avg UT	**0.8488**	**0.9070**	**0.8434**	**0.8543**	0.8468	**0.7041**
Avg IT	0.8473	0.9032	0.8432	**0.8543**	**0.8729**	0.6913

5.1.3. CNN6 Evaluation

Table 3 summarizes CNN6's performance across the three evaluation settings. The first model exhibits the strongest overall performance and is therefore selected for deployment in the prediction time experiments. It achieves the highest scores in most metrics for both the OT and IT test configurations, including top F1 and MCC values on imbalanced data—key indicators of robust classification in the presence of class imbalance. Additionally, it performs well in terms of TNR, reflecting strong NSR detection.

While the fourth model does not outperform the first in aggregate, it demonstrates greater metric stability, particularly with consistently high TNR values across experiments. This reliability could make it a suitable alternative in deployment contexts where consistent performance is preferred over peak accuracy. However, for the purposes of latency evaluation, the superior average performance of the first model makes it the primary candidate.

Table 3. Performance of CNN6. Bold values indicate the best performance across the compared models.

Experiment	ACC	AUC	TPR	TNR	F1	MCC
OT						
CNN6:1	**0.8468**	**0.9169**	0.9265	0.7671	**0.8581**	**0.7026**
CNN6:2	0.7608	0.8467	0.9826	0.5391	0.8042	0.5821
CNN6:3	0.8087	0.8831	0.7469	**0.8706**	0.7961	0.6223
CNN6:4	0.8285	0.8920	0.8366	0.8203	0.8299	0.6571
CNN6:5	0.7949	0.8838	**0.9884**	0.6015	0.8282	0.6398
UT						
CNN6:1	**0.8861**	**0.9329**	0.9265	**0.8457**	**0.8905**	**0.7748**
CNN6:2	0.7829	0.8548	0.9826	0.5832	0.8190	0.6173
CNN6:3	0.7934	0.8677	0.7473	0.8395	0.7835	0.5894
CNN6:4	0.8249	0.8859	0.8363	0.8135	0.8269	0.6500
CNN6:5	0.8349	0.9060	**0.9884**	0.6814	0.8568	0.7038
IT						
CNN6:1	**0.8960**	**0.9436**	0.9265	**0.8457**	**0.9174**	**0.7775**
CNN6:2	0.8321	0.9034	0.9826	0.5832	0.8794	0.6510
CNN6:3	0.7818	0.8637	0.7469	0.8395	0.8101	0.5692
CNN6:4	0.8279	0.8989	0.8366	0.8135	0.8583	0.6410
CNN6:5	0.8727	0.9363	**0.9884**	0.6814	0.9063	0.7364
Avg						
Avg OT	0.8080	0.8845	**0.8962**	0.7197	0.8233	0.6408
Avg UT	0.8244	0.8895	**0.8962**	**0.7527**	0.8353	0.6671
Avg IT	**0.8421**	**0.9092**	**0.8962**	**0.7527**	**0.8743**	**0.6750**

5.1.4. Performance Comparison under Oversampled Test (OT)

Table 4 shows the average performance of the three 1D-CNN models under the OT configuration. Among them, CNN5 achieves the highest overall performance, outperforming CNN4 and CNN6 across most metrics. It shows stronger capability in detecting NSR, while CNN6 demonstrates slightly better performance in AFib detection. However, the difference in AFib classification between CNN5 and CNN6 is marginal, reinforcing CNN5's position as the most balanced model under the oversampled condition.

5.1.5. Performance Comparison under Undersampled Test (UT)

Table 5 presents the performance comparison of the three 1D-CNN models under the UT configuration. CNN5 achieves the highest average performance across most metrics, confirming its effectiveness in this class-balanced scenario. It shows stronger performance in detecting NSR, while CNN6 slightly outperforms it in AFib detection. Despite this minor trade-off, CNN5 maintains the best overall performance among the models in the UT setting.

5.1.6. Performance Comparison under Imbalanced Test (IT)

Table 6 compares the performance of the three 1D-CNN models under the IT configuration. While CNN5 and CNN6 yield comparable average performance overall, they diverge notably in key metrics. CNN6 achieves a higher

TPR, indicating better sensitivity to AFib episodes, whereas CNN5 excels in TNR, reflecting stronger performance in detecting NSR. Notably, CNN5 records a significantly higher MCC value, suggesting more balanced and reliable classification under class imbalance. Given its superior TNR and MCC, CNN5 is considered the more stable and practically suitable model for deployment in imbalanced scenarios.

Table 4. Performance comparison of CNN models on the OT experiment. Bold values indicate the best performance among models.

Experiment	ACC	AUC	TPR	TNR	F1	MCC
OT						
CNN4:1	**0.7372**	0.8103	0.9130	0.5614	**0.7765**	**0.5068**
CNN4:2	0.6774	0.7821	0.4312	**0.9236**	0.5720	0.4077
CNN4:3	0.7068	0.7991	0.5602	0.8533	0.6564	0.4326
CNN4:4	0.7271	0.7624	0.8032	0.6511	0.7464	0.4597
CNN4:5	0.7060	**0.8407**	**0.9778**	0.4341	0.7688	0.4909
Avg	0.7109	0.7989	0.7370	0.6847	0.7040	0.4595
CNN5:1	0.8302	**0.9195**	**0.9334**	0.7271	0.8461	0.6751
CNN5:2	0.8182	0.8621	0.7704	0.8659	0.8090	0.6393
CNN5:3	0.7849	0.8464	0.8303	0.7395	0.7942	0.5722
CNN5:4	0.8302	0.8839	0.7526	**0.9077**	0.8159	0.6685
CNN5:5	**0.8463**	0.9115	0.9295	0.7631	**0.8581**	**0.7025**
Avg	0.8220	0.8847	0.8432	0.8007	0.8247	0.6515
CNN6:1	**0.8468**	**0.9169**	0.9265	0.7671	**0.8581**	**0.7026**
CNN6:2	0.7608	0.8467	0.9826	0.5391	0.8042	0.5821
CNN6:3	0.8087	0.8831	0.7469	**0.8706**	0.7961	0.6223
CNN6:4	0.8285	0.8920	0.8366	0.8203	0.8299	0.6571
CNN6:5	0.7949	0.8838	**0.9884**	0.6015	0.8282	0.6398
Avg	0.8080	0.8845	0.8962	0.7197	0.8233	0.6408

Table 5. Performance comparison of CNN models on the UT experiment. Bold values indicate the best performance among models.

Experiment	ACC	AUC	TPR	TNR	F1	MCC
UT						
CNN4:1	0.8188	0.9032	0.9136	0.7240	0.8345	0.6494
CNN4:2	0.7119	0.8321	0.4306	**0.9931**	0.5991	0.5126
CNN4:3	0.7713	0.8779	0.5610	0.9816	0.7104	0.5981
CNN4:4	**0.8362**	0.8987	0.8025	0.8698	0.8304	0.6739
CNN4:5	0.8351	**0.9411**	**0.9779**	0.6924	**0.8557**	**0.6994**
Avg	0.7946	0.8906	0.7371	0.8521	0.7660	0.6266
CNN5:1	**0.8696**	**0.9391**	**0.9334**	0.8057	**0.8774**	**0.7453**
CNN5:2	0.8130	0.8595	0.7699	0.8561	0.8046	0.6284
CNN5:3	0.8392	0.9038	0.8310	0.8474	0.8379	0.6785
CNN5:4	0.8594	0.9057	0.7527	**0.9662**	0.8426	0.7359
CNN5:5	0.8629	0.9281	0.9297	0.7961	0.8715	0.7324
Avg	0.8488	0.9070	0.8434	0.8543	0.8468	0.7041
CNN6:1	**0.8861**	**0.9329**	0.9265	**0.8457**	**0.8905**	**0.7748**
CNN6:2	0.7829	0.8548	0.9826	0.5832	0.8190	0.6173
CNN6:3	0.7934	0.8677	0.7473	0.8395	0.7835	0.5894
CNN6:4	0.8249	0.8859	0.8363	0.8135	0.8269	0.6500
CNN6:5	0.8349	0.9060	**0.9884**	0.6814	0.8568	0.7038
Avg	0.8244	0.8895	0.8962	0.7527	0.8353	0.6671

Table 6. Performance comparison of CNN models on the IT experiment. Bold values indicate the best performance among models.

Experiment	ACC	AUC	TPR	TNR	F1	MCC
IT						
CNN4:1	0.8766	0.9404	0.9130	0.7240	0.9227	0.6171
CNN4:2	0.5394	0.6062	0.4312	**0.9931**	0.6019	0.3510
CNN4:3	0.6414	0.7289	0.5602	0.9816	0.7161	0.4290
CNN4:4	0.8160	0.8654	0.8032	0.8698	0.8758	0.5660
CNN4:5	**0.9228**	**0.9810**	**0.9778**	0.6924	**0.9534**	**0.7377**
Avg	0.7592	0.8244	0.7371	0.8522	0.8140	0.5402
CNN5:1	**0.8863**	**0.9508**	**0.9334**	0.8057	**0.9120**	**0.7533**
CNN5:2	0.8020	0.8397	0.7704	0.8561	0.8308	0.6064
CNN5:3	0.8366	0.9046	0.8303	0.8474	0.8651	0.6626
CNN5:4	0.8314	0.8784	0.7526	**0.9662**	0.8492	0.6938
CNN5:5	0.8803	0.9423	0.9295	0.7961	0.9074	0.7401
Avg	0.8473	0.9032	0.8432	0.8543	0.8729	0.6913
CNN6:1	**0.8960**	**0.9436**	0.9265	**0.8457**	**0.9174**	**0.7775**
CNN6:2	0.8321	0.9034	0.9826	0.5832	0.8794	0.6510
CNN6:3	0.7818	0.8637	0.7469	0.8395	0.8101	0.5692
CNN6:4	0.8279	0.8989	0.8366	0.8135	0.8583	0.6410
CNN6:5	0.8727	0.9363	**0.9884**	0.6814	0.9063	0.7364
Avg	0.8421	0.9092	0.8962	0.7527	0.8743	0.6750

5.1.7. Summary of Classification Results

Based on the average performance metrics reported in Tables 4–6, CNN5 emerges as the most effective model overall (see Table 7). It achieves an accuracy of 0.8393, AUC of 0.8983, sensitivity of 0.8432, specificity of 0.8346, F1-score of 0.8481, and MCC of 0.6823. Beyond within-dataset comparisons, CNN5 demonstrated consistent performance across varying test distributions—including oversampled, undersampled, and imbalanced settings—indicating robustness to class prevalence shifts that are common in real-world data.

While this study focuses primarily on assessing deployment feasibility on a constrained edge device, generalizability remains a critical concern. To mitigate the risk of patient leakage, the experiments adopted record-wise data splits with no subject overlap. Future work will include cross-dataset validation using other PhysioNet AF databases (e.g., MIT-BIH AF, LTAF2) to further evaluate the model's robustness and generalization capacity.

Table 7. Summary of the best model CNN5 on Tables 4–6.

Experiment	ACC	AUC	TPR	TNR	F1	MCC
OT	0.8220	0.8847	0.8432	0.8007	0.8247	0.6515
UT	0.8488	0.9070	0.8434	0.8543	0.8468	0.7041
IT	0.8473	0.9032	0.8432	0.8543	0.8729	0.6913
Avg	0.8393	0.8983	0.8432	0.8346	0.8481	0.6823

5.2. System-Level Latency Results

This section presents system-level latency results along two dimensions: (i) end-to-end delay, defined as the time taken to transfer ECG segment files from a client device to the processing platform; and (ii) prediction time, the inference latency of the deployed models. Both types of latency are measured independently for cloud and edge deployments to assess the responsiveness of the proposed framework under different computational environments. The following subsections present detailed results for each platform and latency type.

5.2.1. Cloud-Based End-to-End Delay Results

Table 8 summarizes the minimum and average end-to-end delay observed on the cloud platform, Google Drive. Delay is measured as the time elapsed from the moment a file is transmitted by the source to the moment a confirmation is received after successful upload. As expected, both minimum and average delays increase with file size—that is, with the number of 10-second ECG segments per file.

The shortest delay occurs with the smallest file type (4 segments), with minimum and average values of approximately 1.32 and 1.56 s, respectively. These values include overhead from file encryption, API interaction (Google Drive API), and Python execution time, all of which may slightly inflate the observed delays. While actual network transfer latency may be lower, the measurements reflect the full system-level cost of uploading ECG files to the cloud.

Table 8. End-to-end delay performance on cloud.

File Type	Filesize/Segment	No. of Files	Least Delay (sec)	Average Delay (sec)
4	103 kB	200	1.32646	1.56673
8	200 kB	200	1.40235	1.75070
12	297 kB	200	1.56931	1.75429
16	394 kB	200	1.59211	1.77744
24	587 kB	200	1.76973	1.95286
32	781 kB	200	1.98991	2.39910
64	1556 kB	197	2.68940	2.84242
128	3107 kB	113	4.23594	4.43729

5.2.2. Edge-Based End-to-End Delay Results

Table 9 presents corresponding delay results for the edge device, a Raspberry Pi 3B+. As with the cloud setup, delay increases with file size. The minimum and average delays for the smallest files are approximately 0.019 and 0.38 s, respectively—substantially lower than the cloud platform.

Delay is measured from the moment of data transmission to the receipt of a confirmation message from the edge device. Due to the lack of synchronized clocks between source and destination, the actual transmission delay is likely even shorter. Some delay inflation may also occur due to cryptographic operations and Python module overhead. Nevertheless, the edge-based system consistently outperforms the cloud setup in terms of latency, making it the preferred choice due to its reduced end-to-end latency.

Table 9. End-to-end delay performance on edge.

File Type	Filesize/Segment	No. of Files	Least Delay (sec)	Average Delay (sec)
4	103 kB	200	0.01998	0.03872
8	200 kB	200	0.02540	0.05335
12	297 kB	200	0.03155	0.05464
16	394 kB	200	0.03481	0.05784
24	587 kB	200	0.04700	0.08024
32	781 kB	200	0.063	0.12725
64	1556 kB	197	0.09599	0.19683
128	3107 kB	113	0.19199	0.37797

5.2.3. Cloud-Based Prediction Time Results

Table 10 presents the minimum and average prediction times for the three 1D-CNN models deployed on the Google Colaboratory platform. Each model is evaluated in both regular TensorFlow SavedModel and TensorFlow Lite formats. Prediction time is measured by recording two timestamps—before and after the model inference call—using two timing functions from Python's time module: process_time (CPU-based, labeled as "1") and perf_counter (high-precision or GPU-based, labeled as "2") [38].

The results show that the "2" metrics (GPU-based) consistently yield lower latency than the "1" metrics (CPU-based), as expected due to the higher computing capacity of GPUs. Across most configurations, TensorFlow Lite models exhibit faster inference than their full TensorFlow counterparts, confirming their suitability for deployment in latency-sensitive scenarios.

Notably, some anomalies emerge: the CNN4 Lite model has higher prediction time than the more complex CNN5 and CNN6 models, potentially due to GPU initialization overhead or TensorFlow's internal optimization routines. Similarly, CNN4 shows unexpectedly high values for "Avg2" in its regular format. Despite these irregularities, the general pattern indicates that lower complexity models tend to predict faster under CPU-based metrics, while GPU-based performance can be influenced by additional system factors.

Table 10. Prediction time on cloud.

Model	**Least1 (sec)**	**Avg1 (sec)**	**Std1 (sec)**	**Least2 (sec)**	**Avg2 (sec)**	**Std2 (sec)**
CNN4	0.21869	0.24141	0.05387	0.04076	0.05161	0.04305
CNN4 Lite	0.00035	0.00086	0.00032	0.00025	0.00043	0.00034
CNN5	0.25148	0.26982	0.02645	0.04091	0.04657	0.00671
CNN5 Lite	0.00029	0.00071	0.00051	0.00024	0.00045	0.00020
CNN6	0.28612	0.30734	0.02813	0.04107	0.05041	0.02640
CNN6 Lite	0.00029	0.00072	0.00057	0.00024	0.00049	0.00048

5.2.4. Edge-Based Prediction Time Results

Table 11 presents the prediction time measurements for the Raspberry Pi 3B+ running TensorFlow Lite versions of the three 1D-CNN models. As in the cloud setup, predictions are timed using process_time (“1”) and perf_counter (“2”) to account for variations in CPU load and clock resolution.

The results follow a clearer trend: as model complexity increases, so does prediction time. CNN4 consistently achieves the lowest latency across both minimum and average values, indicating that model simplicity directly translates to faster on-device inference.

When comparing edge and cloud platforms using the TensorFlow Lite results from Tables 10 and 11, the cloud shows markedly lower latency. Prediction times on Google Colaboratory reach sub-millisecond levels (around 0.000N seconds), while the edge device consistently reports higher values (around 0.00N seconds). This highlights the computational advantage of cloud infrastructures, although it comes at the cost of increased end-to-end delay due to data transmission overhead.

Table 11. Prediction time on edge.

Model	**Least1 (sec)**	**Avg1 (sec)**	**Std1 (sec)**	**Least2 (sec)**	**Avg2 (sec)**	**Std2 (sec)**
CNN4 Lite	0.00708	0.00732	0.00026	0.00606	0.00616	0.00008
CNN5 Lite	0.00729	0.00750	0.00018	0.00627	0.00638	0.00010
CNN6 Lite	0.00739	0.00761	0.00023	0.00637	0.00649	0.00016

5.3. Result Interpretation and Implications

The evaluation results across oversampled, undersampled, and imbalanced test configurations consistently identify CNN5 as the most appropriate model for deployment on low-cost, intelligent health monitoring devices. Although CNN6 exhibits higher complexity, it does not translate to superior performance metrics. In fact, CNN5 achieves better overall classification accuracy and faster prediction time, making it the preferred candidate for resource-constrained edge deployment.

While prediction times are generally lower on the cloud platform due to its higher computational capacity, end-to-end delay is substantially shorter on the edge device. Considering both metrics together, the edge platform achieves the lowest total response time, which is critical for real-time health monitoring scenarios. Edge computing offers key advantages such as reduced latency, bandwidth conservation, and decreased network load. In contrast, cloud-based inference, although powerful, introduces longer transmission delays and greater dependency on network conditions. Based on these trade-offs, the results suggest that edge computing is the more suitable architecture for IHMS, particularly when rapid response and real-time processing are required.

Although exact parameter counts and Floating Point Operations (FLOPs)—which represent the total number of arithmetic operations required by a model to perform one prediction—were not calculated, the measured prediction time on the Raspberry Pi 3B+ serves as a practical proxy for both computational cost and energy usage. On embedded devices, shorter prediction time typically correlates with lower power consumption and reduced model complexity. Similarly, the measured end-to-end delay reflects not only communication overhead but also effective inference latency, thereby providing a realistic indicator of the system’s overall efficiency. These metrics together offer a meaningful evaluation of computational feasibility in resource-constrained environments, even without direct FLOP or parameter analysis.

To better illustrate the differences between our proposed approach and prior works, Table 12 compares CNN5 with two representative AFib detection models, EdgeCNN [16] and AFibNet [20]. As shown in the table, EdgeCNN adopts a hybrid edge–cloud configuration with five convolutional blocks, achieving 84–87% accuracy but relying on cloud offloading for heavy computation. AFibNet, in contrast, is a high-capacity 1D-CNN optimized for cloud servers, reaching approximately 99% accuracy with 0.02 s inference time on high-performance hardware. The proposed CNN5 executes entirely on a Raspberry Pi 3B+, achieving approximately 84% accuracy with an average prediction time of ≈ 0.007 s. This comparison highlights a clear design trade-off: while EdgeCNN and AFibNet pursue maximum accuracy through larger, cloud-dependent architectures, CNN5 prioritizes real-time and energy-efficient inference directly on the edge device, where latency and power constraints are critical.

Table 12. Comparative overview of AFib detection models

Model	Deployment	Accuracy (%)	Avg. Prediction Time (s)	Key Characteristics
EdgeCNN [16]	Hybrid Edge–Cloud	84–87	Not reported	Five convolutional blocks; requires cloud offloading
AFibNet [20]	Cloud	≈ 99	0.02 (server)	High-capacity 1D-CNN; cloud-centric inference
Proposed CNN5	Edge (Raspberry Pi 3B+)	≈ 84	0.007	Fully on-device, low-latency, energy-efficient inference

Overall, CNN5 offers a balanced solution that favors deployment feasibility without major sacrifices in performance. Its ability to operate on lightweight hardware with minimal latency supports the viability of continuous, real-time ECG monitoring in edge-based IHMS deployments.

While the present results establish technical feasibility on constrained hardware, clinical operating points will be selected and validated prospectively according to the requirements of the intended workflow.

5.4. Additional Experimental Feedback

This section presents supplementary insights that complement the quantitative evaluation of the proposed AFib detection system. First, a physician-led assessment of ECG image segments provides an external reference point to gauge the interpretability and clinical plausibility of the model's predictions. Second, we reflect on classical ECG feature-based methods—specifically QRS and P-wave detectors—and compare their practical implications with our CNN-based approach. These qualitative perspectives offer additional context for understanding the model's behavior and inform future design considerations for real-time, noise-robust detection systems.

5.4.1. Physicians Evaluation of AFib Detection

To assess the interpretability and practical relevance of the proposed AFib detection system, two physicians were asked to evaluate 240 ECG segments extracted from patient 122 of the LONG TERM AF Database, evenly split into 120 AFib and 120 NSR segments. These segments were converted into standard 25 mm/sec paper-speed ECG images using a modified version of the Python module ecg-plot [39], a format familiar to physicians. The image layout adheres to clinical calibration standards, with horizontal grid lines representing 0.04 s per millimeter (small squares) and 0.2 s per 5 mm (large squares), while vertical scaling corresponds to 10 mm per millivolt (mV), as shown in Figure 3.

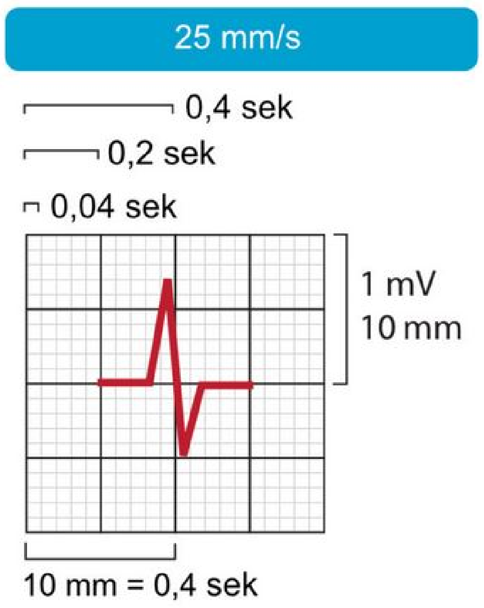

Figure 3. ECG paper-speed 25 mm/sec [40].

To generate these images, x-axis ticks were spaced to create 0.2-second intervals using Matplotlib, with five minor ticks per interval to simulate the smaller 1 mm (0.04 s) boxes. The y-axis was scaled according to ECG amplitude with 0.05 mV increments. Each 10-second segment was split into two 5-second windows and filtered with an 11-point Hamming window to reduce amplitude fluctuations and noise—an adjustment based on physician feedback. An example is shown in Figure 4.

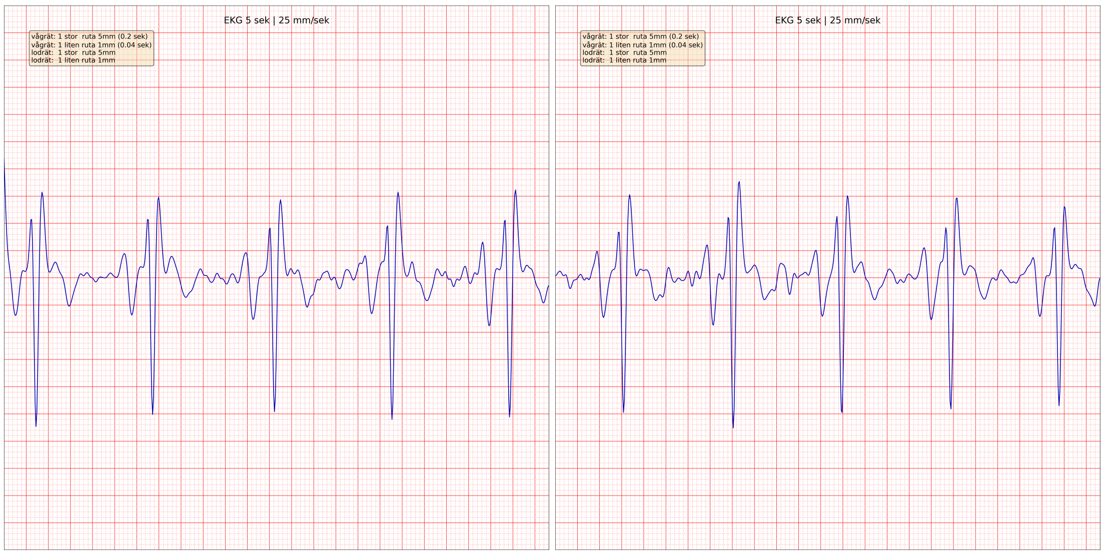
EKG 5 sek | 25 mm/sek
vågrät: 1 stor ruta 5mm (0.2 sek)
vågrät: 1 liten ruta 1mm (0.04 sek)
lodrät: 1 stor ruta 5mm
lodrät: 1 liten ruta 1mm
EKG 5 sek | 25 mm/sek
vågrät: 1 stor ruta 5mm (0.2 sek)
vågrät: 1 liten ruta 1mm (0.04 sek)
lodrät: 1 stor ruta 5mm
lodrät: 1 liten ruta 1mm

Figure 4. Two five second ECG segments (sek = sec).

The physicians independently reviewed the ECG images and classified each segment as AFib, NSR, or Unclear. Table 13 summarizes the classification counts: 105 segments were labeled as AFib, 80 as NSR, and 55 as Unclear. The unclear segments often exhibited significant noise, making clinical interpretation difficult (see Figure 5 for an example).

Table 13. Physicians evaluation of segment distribution.

Total Segments	AFib	NSR	Unclear
240	105	80	55

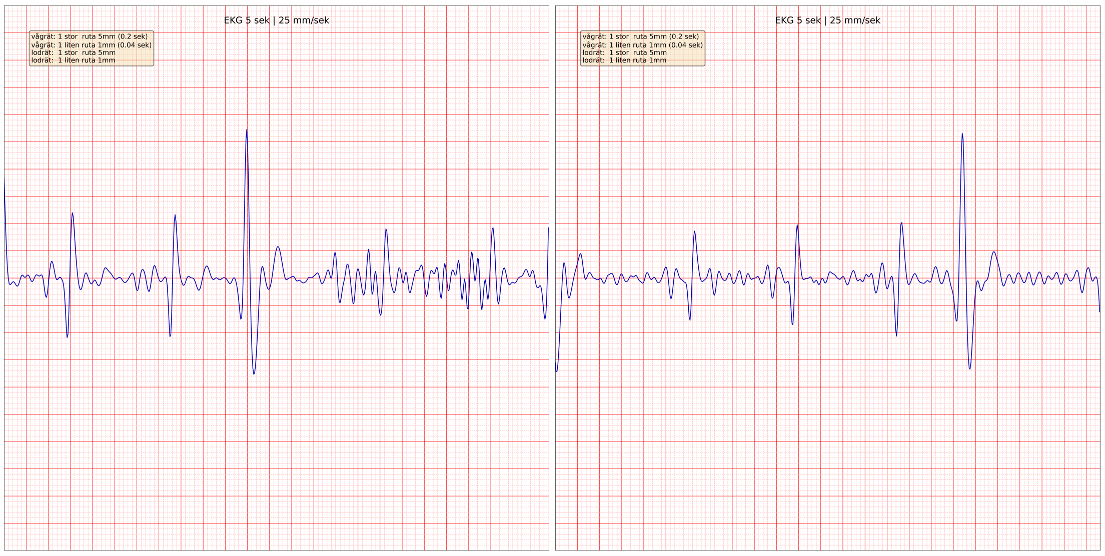
EKG 5 sek | 25 mm/sek
vågrät: 1 stor ruta 5mm (0.2 sek)
vågrät: 1 liten ruta 1mm (0.04 sek)
lodrät: 1 stor ruta 5mm
lodrät: 1 liten ruta 1mm
EKG 5 sek | 25 mm/sek
vågrät: 1 stor ruta 5mm (0.2 sek)
vågrät: 1 liten ruta 1mm (0.04 sek)
lodrät: 1 stor ruta 5mm
lodrät: 1 liten ruta 1mm

Figure 5. Two hard five ECG segments (sek = sec).

To quantify the physicians' performance, a confusion matrix was constructed by treating unclear segments as misclassified relative to the ground truth labels. This conservative approach assumes that failure to provide a confident classification constitutes an error. Based on this, the evaluation yielded 79 true positives (TP), 78 true negatives (TN), 42 false positives (FP), and 41 false negatives (FN). The resulting values of accuracy

($ACC = \frac{TP+TN}{TP+TN+FP+FN}$), sensitivity, or true positive rate ($TPR = \frac{TP}{TP+FN}$), and specificity or true negative rate ($TNR = \frac{TN}{TN+FP}$) are presented in Table 14.

Table 14. Performance of physicians evaluation.

ACC	TPR	TNR
0.6541	0.5031	0.4968

For comparison, the same 240 segments were fed into the three trained CNN models. Their average classification performance is presented in Table 15, where CNN6 achieved the best results overall.

Table 15. Performance of CNN models.

Model	ACC	TPR	TNR
CNN4	0.7958	0.8000	0.7916
CNN5	0.7241	0.6416	0.8066
CNN6	0.8116	0.8166	0.8066

The comparison highlights several key observations. First, the relatively low physician accuracy reflects the intrinsic difficulty of the selected segments, many of which contain noise or ambiguous signal patterns. While CNN6 outperforms the physicians on this set, this should not be interpreted as superiority of the algorithm. It is important to note that the physicians did not conduct a collaborative review or consensus-based labeling due to time constraints, which may have impacted the final classifications. Furthermore, although the CNNs were trained on the same database, subtle variations in segment selection and pre-processing could still affect model generalization.

Overall, this evaluation offers complementary insights into model performance relative to expert judgment and demonstrates the potential of the proposed models to serve as decision-support tools in real-world clinical workflows.

5.4.2. Role of QRS and P-Wave Detectors

This work employs an approach that transforms filtered and normalized ECG signal data into fixed-length segments directly fed into convolutional neural networks (CNNs). The main advantage of this method lies in its simplicity: it bypasses complex feature engineering by relying on the CNNs for automatic feature extraction. However, this benefit comes at the cost of increased computational complexity. Moreover, insufficient noise suppression in the raw ECG signal can negatively impact detection accuracy. While this raw-signal-driven strategy has shown promise, it is not the only viable method for AFib detection.

Alternative approaches typically extract handcrafted features such as RR intervals or assess P-wave visibility—both of which are clinically significant for identifying AFib. RR intervals are computed by detecting R-peaks using QRS detectors, while the presence or absence of P-waves is assessed using specialized P-wave detectors. These methods introduce additional computational steps but may offer better robustness to noise when properly implemented. Given the potential value of these approaches, several state-of-the-art real-time detectors were reviewed for both QRS complexes and P-waves.

Several real-time QRS detectors have been developed, offering high sensitivity on benchmark datasets such as the MIT-BIH Arrhythmia Database. For instance, the detector in [41] reports a sensitivity of 99.88%, while [33,42] achieve 99.74% and 99.75%, respectively. These detectors share common principles, including adaptive thresholding and noise suppression via filtering. The detectors in [33,41] incorporate a learning phase followed by detection: the former uses ten seconds of clean ECG data to initialize thresholds and applies detection every three seconds, while the latter adapts thresholds continuously using just two seconds of data. Noise filtering also varies—Ref. [41] uses Stationary Wavelet Transform, whereas [33] combines band-pass filtering and differentiation. The detector by Christov [42] applies three moving average filters and merges three adaptive thresholds into a unified rule, without a distinct learning phase. While [41] offers the highest reported sensitivity, it requires longer clean ECG segments for initialization. In contrast, Ref. [33] operates effectively with shorter input signals and also reports detection delay, which can improve R-peak localization—making it a better fit for short segments such as the 10-second windows used in this study.

Two real-time P-wave detectors were also reviewed. The detector in [43] identifies P-wave slopes using a fixed search window—half the RR-interval—between two R-peaks, achieving a sensitivity of 97.58%. In contrast, the detector in [44] employs a discrete wavelet transform, using specific frequency bands (third and fourth scales) and adaptive thresholds to detect P-waves, reporting a slightly higher sensitivity of 98.5%. While both approaches rely on thresholding, they differ in strategy: the former combines fixed windows with slope detection, while the latter extracts signal features through wavelet decomposition. These detectors demonstrate strong performance, yet they rely heavily on effective preprocessing and are more vulnerable to noise, especially given the low amplitude and frequency of P-waves.

Although this work initially considered incorporating features like RR intervals and P-wave visibility, the final design focused exclusively on raw ECG segments. The goal was to better understand the strengths and limitations of a purely data-driven approach using CNNs. However, one critical insight from this analysis is that robustness to noise remains a limiting factor for all AFib detection systems. While raw signal analysis preserves maximal information, it is highly susceptible to signal degradation. RR-interval–based detection is more noise-tolerant since R-peaks are typically the most prominent features in ECG signals, but its specificity is lower due to shared rhythm patterns among multiple arrhythmias. Similarly, P-wave absence is a key marker for AFib, yet detecting P-waves in noisy or overlapping waveforms remains a non-trivial task.

Ultimately, effective noise suppression is the most essential requirement for any feature-based or deep learning method aiming at real-time AFib detection. While this study chose to explore the trade-offs of using raw signals without explicit feature extraction, future research may benefit from hybrid models that combine learned features with robust, low-level signal descriptors such as RR intervals or P-wave metrics.

5.5. Limitations and Outlook

A limitation of the presented results is that the ECG preprocessing pipeline, particularly the noise removal step, does not fully suppress high-frequency EMG artifacts in certain recordings. While the first-order Butterworth bandpass filter (5–15 Hz) effectively removes baseline wander and emphasizes the QRS complex, residual noise remains in some segments of the Long-Term AF Database. These recordings were collected using Holter devices under uncontrolled daily-life conditions, and the exact recording environments are not documented, resulting in variable levels of motion and muscle noise across subjects.

As clarified in Section 3.2, the selected 5–15 Hz passband follows classical QRS-focused filtering practices and was chosen to preserve computational simplicity for real-time execution on the Raspberry Pi 3B+. However, this narrow bandwidth inevitably attenuates portions of the P- and T-waves and provides only limited suppression of strong EMG bursts. This limitation became particularly visible when converting ECG segments to images for physician evaluation (Section 5.4.1), especially for noisy records such as record 122.

While more advanced denoising techniques—such as discrete wavelet transforms, adaptive filtering, or empirical mode decomposition—can provide stronger attenuation of high-frequency artifacts, these approaches typically incur higher computational and memory demands than feasible for the embedded hardware used in this work. Further refinement of the noise removal technique or the adoption of more capable edge processors may therefore be necessary for applications targeting diagnostic-grade ECG analysis; however, these limitations do not affect the validity of the comparative findings reported here. Besides, further improvements may be possible through CNN architecture refinement, including adjustments to pooling configurations or layer design. Additionally, the use of SMOTE to balance class distributions may distort the temporal structure of ECG signals, when synthetic segments disrupt natural rhythm transitions or subtle waveform dynamics. Although we found no visual artifacts or performance degradation in our evaluation, future work should investigate more time-aware augmentation strategies to preserve physiological dynamics.

The limitations of the end-to-end delay experiments on the edge device include the absence of Network Time Protocol (NTP) synchronization. Because the clocks of the data source and the Raspberry Pi 3B+ were not synchronized, the reported latency values may include a small timing offset. Given that the measured delays are in the millisecond range, this offset could influence absolute latency values; however, since both edge and cloud tests were conducted under identical unsynchronized conditions, the relative performance comparison remains reliable.

The edge testbed setup also diverges from a real-world deployment scenario, as it uses a cable network connection and a fixed data source. For the cloud-based experiment, delay was measured via Google Drive and Google Colaboratory, which are not representative of production-grade IHMS deployments. More realistic infrastructure and synchronized timing will be implemented in future testbeds. Google Drive is unlikely to be used in a real IHMS deployment, and more suitable cloud service configurations may offer improved delay measurement. However, Google Drive serves a practical purpose in this study because it can be mounted directly on Google Colaboratory. More representative network configurations and synchronized timing will be considered when scaling up the testbed in subsequent studies.

The limitation of the prediction-time experiment on the cloud (Google Colaboratory) is that the GPU is not properly initialized before prediction begins, which may cause the measured prediction times on the cloud to be inaccurate. While this initialization overhead does not change the broader conclusions, a more rigorous GPU-initialization protocol is needed for precise benchmarking.

The reported latency values, especially for cloud-based inference, may be influenced by network conditions such as uplink bandwidth, congestion, and routing variability. In contrast, the edge inference pipeline operates locally on-device and is therefore unaffected by such network fluctuations. While both testbeds (edge and cloud) were evaluated under stable Ethernet conditions for consistency, we acknowledge that latency measurements in more variable wireless networks may yield higher variance, particularly for cloud scenarios. Future evaluations will incorporate emulation of degraded or congested network conditions to better characterize performance trade-offs across realistic deployment environments.

Another limitation of the present study concerns the clinical validation process. The evaluation was performed by two expert annotators and was primarily designed as a feasibility-oriented assessment to verify that the model's predictions align with expert judgment. The inclusion of an Unclear category allowed the experts to flag diagnostically ambiguous ECG segments, which are valuable for identifying borderline rhythm patterns that may also challenge automated classifiers. In future work, this evaluation can be expanded to include additional annotators and patient recordings under blinded conditions. Agreement analysis may then be performed using Cohen's κ (for binary and multiclass cases), Fleiss's κ(for more than two raters), and percent agreement to quantify inter-rater reliability, with a binary κ also computed on determinate AFib and NSR labels for comparability.

6. Conclusions

This study presents the design and evaluation of a low-cost Intelligent Health Monitoring System (IHMS) for atrial fibrillation (AFib) detection using ECG signals. Three one-dimensional convolutional neural networks (1D-CNNs) with varying architectural complexity were developed and tested for deployment on both edge and cloud platforms. Among the three models, CNN5 demonstrated the most favorable balance between detection performance and computational efficiency. When evaluated across over-sampled, under-sampled, and imbalanced test configurations, CNN5 achieved an accuracy of 83.93%, AUC of 89.83%, sensitivity of 84.32%, specificity of 83.46%, F1-score of 84.81%, and MCC of 68.23%. These results confirm CNN5 as the most suitable model for low-resource environments requiring real-time, on-device inference.

System-level latency analysis revealed that while cloud execution offers faster raw prediction times, the overall response time—accounting for both prediction and communication delays—is significantly shorter on the edge. The Raspberry Pi 3B+ demonstrated end-to-end delays ranging from 0.019 to 0.377 s, compared to 1.32 to 4.43 s on the cloud platform. These results suggest that edge computing is a more viable architecture for IHMS applications where low latency, autonomy, and privacy are essential. Although high-capacity models like AfibNet achieve superior accuracy in data center settings, they are not compatible with real-time, low-power operation. By contrast, CNN5 enables fully on-device inference, making it a strong candidate for deployment in portable health monitoring systems.

The reported classification performance should be viewed through the lens of technical feasibility rather than clinical validation. Current medical guidelines do not enforce fixed performance thresholds for AFib detection; rather, requirements vary by use case and workflow. Accordingly, this work does not make clinical claims, but demonstrates that real-time AFib detection is feasible on low-cost hardware. Future work will include prospective evaluation using receiver-operating-characteristic (ROC) analysis, reporting confidence intervals, and selecting thresholds tailored to intended clinical contexts.

Although this study demonstrates deployment feasibility, further optimization is needed. Potential enhancements include model pruning, quantization, and hardware-aware architecture search to improve performance without increasing complexity. Improvements in noise suppression and real-time feature extraction can also benefit model robustness, especially in less controlled environments. Additional efforts will focus on expanding to wearable and mobile platforms, cross-dataset generalization, and broader physician-in-the-loop assessments under blinded conditions. Moreover, latency benchmarking will be strengthened through synchronized timing and more realistic network emulation to better characterize system behavior under operational constraints. Together, these extensions will support the development of a more accurate, robust, and clinically aligned real-time AFib detection system.

Author Contributions: P.N.: methodology, software, investigation, experiments, writing—original draft preparation; H.F.: conceptualization, supervision, methodology; M.L.: conceptualization, supervision, methodology; O.C.: writing—review and editing; T.S.: writing—review and editing; M.V.: writing—review and editing.

Funding: This research was funded by THCS European program via RENEW Project, and by the Excellence in Production Research Framework through XPRES (Excellence in Production Research), and by R2Microgrid project under the RESILIENT competence center, financed by the Swedish Energy Agency and co-financed by Mälardalen University and industrial partners.

Data Availability Statement: The data used in this study are publicly available from the PhysioNet repository (https://physionet.org), including datasets containing atrial fibrillation and normal sinus rhythm ECG recordings used for model training and evaluation.

Conflicts of Interest: The authors declare no conflict of interest.

Use of AI and AI-Assisted Technologies: No generative AI tools were used for content generation or data analysis. AI was employed solely for language editing and proofreading support.

Abbreviations

AFib	Atrial Fibrillation
PAF	Paroxysmal Atrial Fibrillation
ECG	Electrocardiogram
BLW	Baseline Wander
PLI	Powerline Interference
EMG	Electromyogram
NSR	Normal Sinus Rhythm
CNN	Convolutional Neural Network
1D	One Dimensional
AUC	Area Under the Curve
TPR	True Positive Rate (Sensitivity)
TNR	True Negative Rate (Specificity)
MCC	Matthews Correlation Coefficient
IHMS	Intelligent Health Monitoring System

References

1. Kotecha, D.; Piccini, J.P. Atrial Fibrillation in Heart Failure: What Should We Do? *Eur. Heart J.* **2015**, *36*, 3250–3257.
2. Chugh, S.S.; Havmoeller, R.; Narayanan, K.; et al. Worldwide Epidemiology of Atrial Fibrillation: A Global Burden of Disease 2010 Study. *Circulation* **2014**, *129*, 837–847.
3. Morillo, C.; Banerjee, A.; Perel, P.; et al. Atrial Fibrillation: The Current Epidemic. *J. Geriatr. Cardiol.* **2017**, *14*, 195–203.
4. Rizwan, A.; Zoha, A.; Mabrouk, I.; et al. A Review on the State of the Art in Atrial Fibrillation Detection Enabled by Machine Learning. *IEEE Trans. Biomed. Eng.* **2020**, *14*, 219–239.
5. Xie, L.; Li, Z.; Zhou, Y.; et al. Computational Diagnostic Techniques for Electrocardiogram Signal Analysis. *Sensors* **2020**, *20*, 6318.
6. Ansari, Y.; Mourad, O.; Qaraqe, K.; et al. Deep Learning for ECG Arrhythmia Detection and Classification: An Overview of Progress for Period 2017–2023. *Front. Physiol.* **2023**, *14*, 1246746.
7. Mant, J.; Fitzmaurice, D.A.; Hobbs, F.R.; et al. Accuracy of Diagnosing Atrial Fibrillation on Electrocardiogram by Primary Care Practitioners and Interpretative Diagnostic Software: Analysis of Data from Screening for Atrial Fibrillation in the Elderly (SAFE) Trial. *BMJ* **2007**, *335*, 380.
8. Kashou, A.H.; May, A.M.; Noseworthy, P.A.; et al. ECG Interpretation Proficiency of Healthcare Professionals. *Curr. Probl. Cardiol.* **2023**, *48*, 101924.
9. Dash, S.; Chon, K.; Lu, S.; et al. Automatic Real Time Detection of Atrial Fibrillation. *Ann. Biomed. Eng.* **2009**, *37*, 1701–1709.
10. Shashikumar, S.P.; Shah, A.J.; Li, Q.; et al. A Deep Learning Approach to Monitoring and Detecting Atrial Fibrillation Using Wearable Technology. In Proceedings of the 2017 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI), Orlando, FL, USA, 16–19 February 2017; pp. 141–144.
11. Nemati, S.; Ghassemi, M.M.; Ambai, V.; et al. Monitoring and Detecting Atrial Fibrillation Using Wearable Technology. In Proceedings of the 2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Orlando, FL, USA, 16–20 August 2016; pp. 3394–3397.
12. Kiranyaz, S.; Ince, T.; Abdeljaber, O.; et al. 1-D Convolutional Neural Networks for Signal Processing Applications. In Proceedings of the ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, 12–17 May 2019; pp. 8360–8364.
13. Ebrahimi, Z.; Loni, M.; Daneshtalab, M.; et al. A Review on Deep Learning Methods for ECG Arrhythmia Classification. *Expert Syst. Appl. X* **2020**, *7*, 100033.
14. Pourbabaee, B.; Roshtkhari, M.J.; Khorasani, K. Deep Convolutional Neural Networks and Learning ECG Features for Screening Paroxysmal Atrial Fibrillation Patients. *IEEE Trans. Syst. Man Cybern. Syst.* **2018**, *48*, 2095–2104.
15. Andersen, R.S.; Peimankar, A.; Puthusserypady, S. A Deep Learning Approach for Real-Time Detection of Atrial Fibrillation. *Expert Syst. Appl.* **2019**, *115*, 465–473.
16. Yu, J.; Fu, B.; Cao, A.; et al. EdgeCNN: A Hybrid Architecture for Agile Learning of Healthcare Data from IoT Devices. In

Proceedings of the 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), Singapore, 11–13 December 2018; pp. 852–859.

17. Lee, Y.J.; Kim, H.; Lee, J.; et al. Artificial Intelligence of Things Wearable System for Cardiac Disease Detection. In Proceedings of the 2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), Hsinchu, Taiwan, 18–20 March 2019; pp. 67–70.
18. Ma, F.; Zhang, J.; Liang, W.; et al. Automated Classification of Atrial Fibrillation Using Artificial Neural Network for Wearable Devices. *Math. Probl. Eng.* **2020**, *2020*, 9159158.
19. Hou, D.; Raymond Hou, M.; Hou, J. ECG Beat Classification on Edge Device. In Proceedings of the 2020 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 4–6 January 2020; pp. 1–4.
20. Tutuko, B.; Nurmaini, S.; Tondas, A.E.; et al. AFibNet: An Implementation of Atrial Fibrillation Detection with Convolutional Neural Network. *BMC Med. Inform. Decis. Mak.* **2021**, *21*, 216.
21. Xia, Y.; Wulan, N.; Wang, K.; et al. Detecting Atrial Fibrillation by Deep Convolutional Neural Networks. *Comput. Biol. Med.* **2018**, *93*, 84–92.
22. Khan, F.; Yu, X.; Yuan, Z.; et al. ECG Classification Using 1-D Convolutional Deep Residual Neural Network. *PLoS One* **2023**, *18*, e0284791.
23. Ahmed, A.A.; Ali, W.; Abdullah, T.A.; et al. Classifying Cardiac Arrhythmia from ECG Signal Using 1D CNN Deep Learning Model. *Mathematics* **2023**, *11*, 562.
24. Mallikarjunamallu, K.; Khasim, S. Arrhythmia Classification Using Noise Filtering and 1D CNN. *Trait. Signal* **2024**, *41*.
25. Gao, J.; Li, Y.; Chen, M.; et al. Signal Reconstruction Method Based CNN for Atrial Fibrillation Detection. In Proceedings of the 2024 17th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), Beijing, China, 19–21 October 2024; pp. 1–6.
26. Gao, J.; Li, Y.; Chen, M.; et al. Efficient Transformation of ECG Signals from 1-D to 2-D for Atrial Fibrillation Detection Using Deep Learning. *Signal Image Video Process.* **2025**, *19*, 695.
27. Elmir, Y.; Himeur, Y.; Amira, A. Federated Learning with Gramian Angular Fields for Privacy-Preserving ECG Classification on Heterogeneous IoT Devices. *arXiv* **2025**, arXiv:2511.03753.
28. Ahmad, F.; Zafar, S. SoC-Based Implementation of 1-D Convolutional Neural Network for 3-Channel ECG Arrhythmia Classification via HLS4ML. *IEEE Embed. Syst. Lett.* **2024**, *16*, 429–432.
29. Guhdar, M.; Mohammed, A.O.; Mstafa, R.J. Advanced Deep Learning Framework for ECG Arrhythmia Classification Using 1D-CNN with Attention Mechanism. *Knowl.-Based Syst.* **2025**, *315*, 113301.
30. Petrutiu, S.; Sahakian, A.V.; Swiryn, S. Abrupt Changes in Fibrillatory Wave Characteristics at the Termination of Paroxysmal Atrial Fibrillation in Humans. *EP Europace* **2007**, *9*, 466–470.
31. Goldberger, A.L.; Amaral, L.A.; Glass, L.; et al. PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals. *Circulation* **2000**, *101*, e215–e220.
32. Xie, C.; McCullum, L.; Johnson, A.; et al. *Waveform Database Software Package (WFDB) for Python*, Version 4.0.0; PhysioNet: Cambridge, MA, USA, 2021.
33. Pan, J.; Tompkins, W.J. A Real-Time QRS Detection Algorithm. *IEEE Trans. Biomed. Eng.* **1985**, *BME-32*, 230–236.
34. The Functional API. Available online: https://keras.io/guides/functional_api/ (accessed on 10 March 2026).
35. Chawla, N.V.; Bowyer, K.W.; Hall, L.O.; et al. SMOTE: Synthetic Minority Over-Sampling Technique. *J. Artif. Intell. Res.* **2002**, *16*, 321–357.
36. Random-Under-Sampler. Available online: https://tinyurl.com/3r2ap5c5 (accessed on 10 March 2026).
37. Introduction to Google Drive API. Available online: https://tinyurl.com/4cd5847s (accessed on 10 March 2026).
38. Simpson, C.; Jewett, J.J.; Turnbull, S.J.; et al. PEP 418—Add Monotonic Time, Performance Counter, and Process Time Functions. Available online: https://peps.python.org/pep-0418/ (accessed on 10 March 2026).
39. ecg-plot 0.2.8. Available online: https://pypi.org/project/ecg-plot/ (accessed on 10 March 2026).
40. Lär Dig EKG-Tolkning Och Mycket Mer—Sveriges Mest Använda EKG-Bok. Available online: https://ekg.nu/ (accessed on 10 March 2026).
41. Kalidas, V.; Tamil, L. Real-Time QRS Detector Using Stationary Wavelet Transform for Automated ECG Analysis. In Proceedings of the 2017 IEEE 17th International Conference on Bioinformatics and Bioengineering (BIBE), Washington, DC, USA, 23–25 October 2017; pp. 457–461.
42. Christov, I.I. Real Time Electrocardiogram QRS Detection Using Combined Adaptive Threshold. *Biomed. Eng. Online* **2004**, *3*, 28.
43. Chatterjee, H.; Gupta, R.; Mitra, M. Real Time P and T Wave Detection from ECG Using FPGA. In Proceedings of the 2nd International Conference on Computer, Communication, Control and Information Technology (C3IT-2012), Hooghly, India, 25–26 February 2012; pp. 840–844.
44. Sovilj, S.; Jeras, M.; Magjarevic, R. Real Time P-Wave Detector Based on Wavelet Analysis. In Proceedings of the 12th IEEE Mediterranean Electrotechnical Conference (MELECON), Dubrovnik, Croatia, 12–15 May 2004; Volume 1, pp. 403–406.