Scalable, Multiplatform, and Autonomous ECG Processor Supported by AI for Telemedicine Center

Filip Plesinger[1], Adam Ivora[1], Eniko Vargova[1], Radovan Smisek[1], Jan Pavlus[1], Zuzana Koscova[1], Petr Nejedly[1], Veronika Bulkova[2], Roman Kozubik[2], Josef Halamek[1], Pavel Jurak[1]

[1]Institute of Scientific Instruments of the CAS, Czechia
[2]Medical Data Transfer, Czechia

Abstract

***Background**: Wearable devices play an essential role in the early diagnosis of heart diseases. However, effective management of long-term ECG measurements (1-3 weeks) by a telemedicine center (TMC) requires specifically designed software.*

***Method**: We used the multiplatform framework .NET to build the application. Deep-learning models for QRS detection, classification, and rhythm analysis were trained in the PyTorch framework; models were trained using data from Medical Data Transfer, s. r. o., Czechia (N=73,450 and 12,111). The ONNX runtime libraries were used for model inference, including acceleration by graphic cards*

***Results**: The pre-production benchmark (recordings of 82 patients) showed a mean accuracy of 0.97 ± 0.04 for QRS detection and classification into three classes; it also showed a mean accuracy of 0.97 ± 0.03 for rhythm classification into seven classes.*

***Conclusion**: The presented software is a fully automated, multiplatform, and scalable back-end application to process incoming ECG data in the TMC. Although it is not freely accessible, we are open to considering processing ECG data for research and strictly non-commercial purposes.*

1. Introduction

Early detection of cardiac disorders usually improves treatment efficiency. Wearables (bands, watches), episode recorders, or ECG holters provide ECG recordings, usually transmittable using a wireless connection to cloud services. However, a telemedicine center (TMC) that uses several kinds of these devices and simultaneously aims for nearly real-time analysis represents a specific challenge for software development. In this paper, we present an autonomous ECG processor – a stand-alone back-end application for use in an ecosystem of a TMC.

2. Data

Presented software contains two already developed deep-learning models – the first for QRS detection/classification [1] and the second model for rhythm classification. Used data originated from a private dataset (MDT, Brno, Czechia), consisting of 12,111 ECG recordings with 619,681 QRS annotations (length 45 seconds, 200Hz sampling) and 73,450 recordings accompanied by rhythm annotations (length 30 s, 200 Hz sampling).

The software presented in this paper was verified using a set of 82 files from various devices (episode recorders, Holter recorders, wearables). The set consisted of private (MDT) and public (PhysioNet Challenge 2015 [2]) data. Sampling frequencies varied from 125 to 256 Hz (med. 250 Hz, IQR 0 Hz) in lengths from 30 to 17,115 seconds (1,057±2,915). Table 1 shows further dataset description, grouped by data origin. Recordings were semi-automatically annotated using the SignalPlant software [3].

Table 1. Verification (benchmark) dataset by data origin. PAC-premature atrial contraction, PVC-premature ventricular contraction, VT-ventricular tachycardia (sustained), NSVT-non-sustained ventricular tachycardia, SVT-supraventricular tachycardia run, AFIB-atrial fibrillation, AVBII-2nd degree AV-block. PC2015-CinC/PhysioNet Challenge 2015, MDT-Medical data transfer, Brno, Czechia.

Dataset	QRS count (Files)	Device	Pathologies
MDT	110,266 (67)	Faros, Getemed, Vitaphone	PAC, PVC, VT, NSVT, SVT, AVBII, AFIB, Pauses, Noise
PC2015 [2]	1,028 (15)	Unknown	PAC, PVC, VT, NSVT, Noise

3. Method

The presented software is designed as a multiplatform command line tool. It allows execution in several modes; the one specific to the TMC is to wait at a specific folder (shared with other software instances running on other computers) and wait for incoming ECG recordings (usually 30-60 s or 1 hour long). Then each waiting file is processed (Fig.1), and the solver waits again.

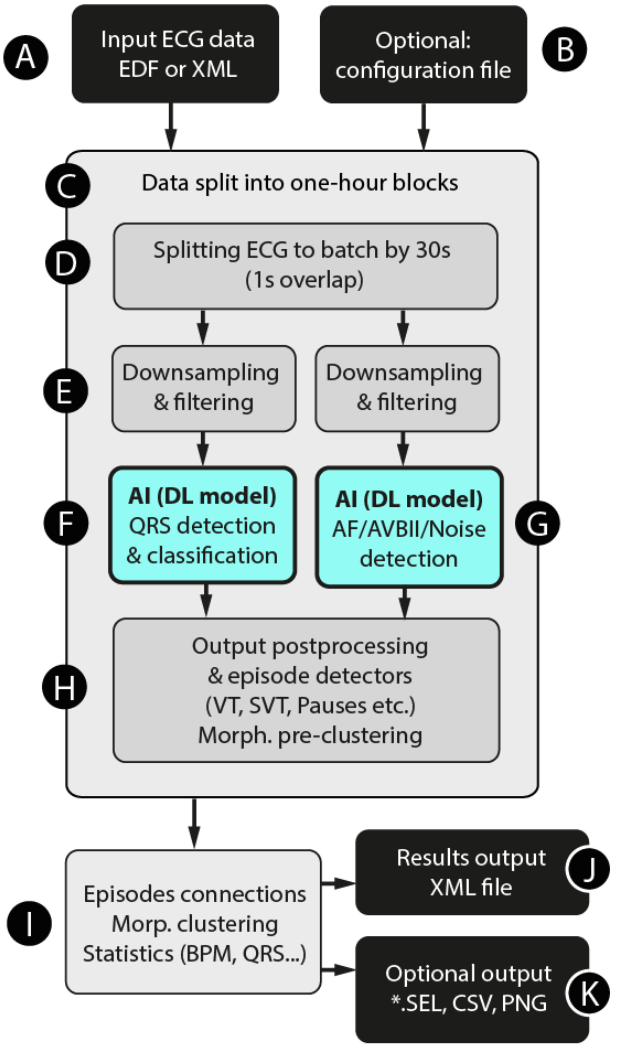

Figure 1. Simplified software flowchart. Input ECG signal (A) is loaded, optionally with a configuration file (B). The signal is split into one-hour chunks (C); each chunk is split into a batch with 30 s long signals (D). The signal is filtered (0.5-45 Hz), down-sampled to 100 and 200 Hz (E), and fed into two deep-learning models (F, G). Model outputs are post-processed (H), and rhythm episodes are connected between one-hour blocks (I). Finally, results are exported as an XML file and, optionally, as SignalPlant [3] mark file (*.sel) or others.

3.1 Deep-learning models

The “heart” of the software is formed by two deep-learning models, both working in 30 s blocks. Input to both models is filtered (0.5-45 Hz) and standardized. The first one detects and classifies QRS complexes into three classes [1] (Fig.1F). The second model (ResNet architecture) determines heart rhythm (AFIB, AVBII, or noise) in the group. Both models are linked to the application using the Open Neural Network Exchange (ONNX) file format; therefore, they can be updated without significant changes to the core application.

3.2 Output postprocessing

Other rhythm disorders are computed during the post-processing stage when information is fused from both models: VT, NSVT, pauses, SVT runs or blocks, and PVCs or PACs in various patterns such as couplets, triplets, bigeminies, etc. The software also provides clustering by QRS morphology (Fig.1H and I).

However, results from each one-hour block must be connected to surrounding one-hour blocks. Also, the average QRS from each morphological group is used for final morphological clustering over all one-hour blocks. For each one-hour block, a heart rate descriptive statistic is also included (Fig.1I).

Finally, the results are exported as an XML file containing all rhythm episodes, a description of present morphologies and their belonging to a specific QRS class, and a list of QRS complexes with their class and morphological group. (Fig.1J). Optionally, episodes and QRS marks can be exported in SignalPlant [3] format for annotation marks (*.sel) or as CSV or PNG files containing raw model output (Fig.1K).

We used C# programming language with the .NET framework to build the application (ver. 5.0). Deep-learning inference is provided by Microsoft.ML.ONNX libraries (ver. 1.11.0).

We used Python 3.8 with Pandas [4] and Numpy [5] packages for model training codes; the PyTorch framework [6] was used to train deep-learning models.

3.3. Verification with benchmark

We built an automated pipeline to verify software after each commit to the local GitLab repository. We used Python 3.8, Pandas [4], Numpy [5], and Scikit-learn [7] packages for benchmark development. The benchmark checks the quality of QRS detection and classification, other rhythm disorders scores, and noise detection scores.

Since the target deployment is in the TMC, we also needed to estimate computing time in different processing scenarios. The Wilcoxon signed-rank pair test was used for statistical testing.

4. Results

We developed a fully automated, back-end application classifying ECG signals. The latest stable version deployed in TMC (0.3.110) worked using a cluster of one physical and four virtual computers. These computers shared a single network folder for incoming data and have run without crashes since November 2021. The current load is approximately 400x 1-hour ECG recordings per computing hour, occupying one-third of the total computational capacity. In addition, other software instances process shorter recordings (30 s) from wearables such as smartwatches or bands.

4.1. QRS Complexes

The benchmark showed a mean weighted QRS detection/classification accuracy of 0.97 ± 0.044 in 82 ECG files. Table 2 shows specific scores for each of the separate QRS classes Normal, PAC and PVC.

Table 2. Mean accuracy of QRS detection and classification of QRS complexes. PAC - premature atrial contraction, PVC – premature ventricular contraction. (Overall accuracy is weighted).

QRS type	Count	Accuracy
Normal	95,399	0.980 ± 0.035
PVC	13,384	0.918 ± 0.191
PAC	2,379	0.928 ± 0.185
Overall	111,162	0.972 ± 0.044

4.2. Rhythm classification

The automated benchmark pipeline analyzed the classification performance of seven rhythm groups (Tab.3).

Table 3. Performance of rhythm classification into seven groups. AFIB-atrial fibrillation, AVB(II)-2nd degree AV block (type I+II), VT-sustained ventricular tachycardia, NSVT-non-sustained ventricular tachycardia, SVT-supraventricular tachycardia.

Rhythm	In files	Total seconds	Accuracy
AFIB	21	27,507	0.921 ± 0.206
AVB(II)	7	653	0.932 ± 0.216
PAUSE	12	254	0.997 ± 0.017
VT	9	59	0.991 ± 0.031
NSVT	11	102	0.988 ± 0.037
SVT	10	520	0.996 ± 0.017
NOISE	15	1,707	0.952 ± 0.119
Overall	82	30,802	0.968 ± 0.030

Overall average accuracy reached 0.968 ± 0.030 with a minimal value of 0.921 ± 0.216 for AF; maximal accuracy was reached for pauses (0.997). The highest standard deviation was found in AVBII and AF classes.

The software also detects other pathological rhythms tightly linked to the QRS class as PVC couplets, triplets, bigeminies, etc. These other pathological classes are not checked with benchmark tests since they are already covered during the test of QRS complexes. The only exceptions are VT, NSVT, and SVT, presented in Tab.3.

4.3. Processing Time

During the benchmark test, we also monitored computing performance. The hardware configuration for the test was an HP Z6 G4 workstation with CPU Intel® Xeon ® Silver 4116 CPU at 2.1 GHz, 64 GB of RAM, and running on the 64-bit operating system Windows 10. We did the benchmark twice; for the first, deep-learning inference was done only using CPU; the other run used one GPU Nvidia GeForce GTX 1080 Ti for inference. Processing times were evaluated for three groups of files from the benchmark dataset: 30-60 seconds (N=26), 60-120 seconds (N=35), and one hour (N=10).

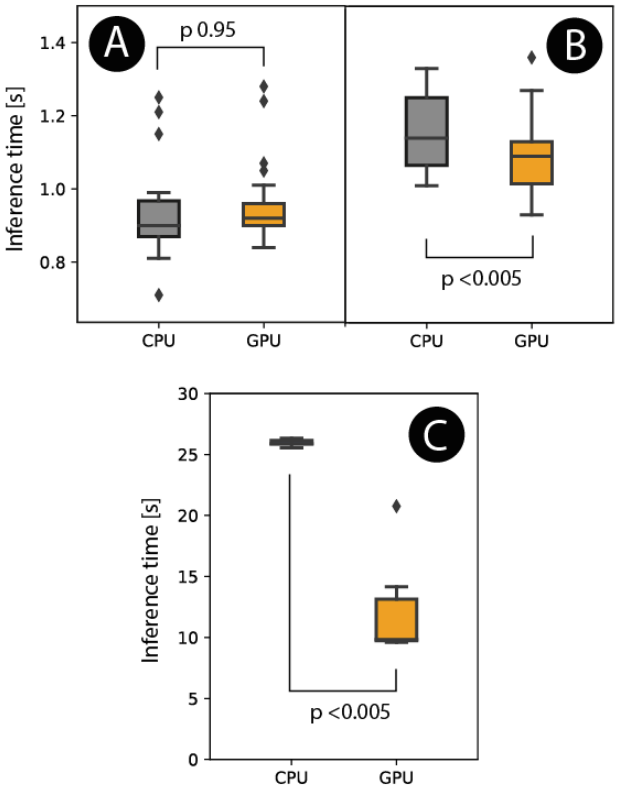

Figure 2. The effect of using GPU (orange) or CPU (gray) for deep-learning inference in three benchmark subsets by file length: 30-60 seconds (A, N=26), 60-120 seconds (B, N=35), and 3,600 seconds (C, N=10).

Fig. 2A shows that in shorter files (30-60 seconds), GPU does not show a lower processing time than the CPU (p=0.95). However, in slightly longer files (Fig.2B), using GPU lowers computing times (p=0.003). In long files (3,600 seconds, Fig.2C), the total computing time is nearly 2.5 times shorter (p=0.003).

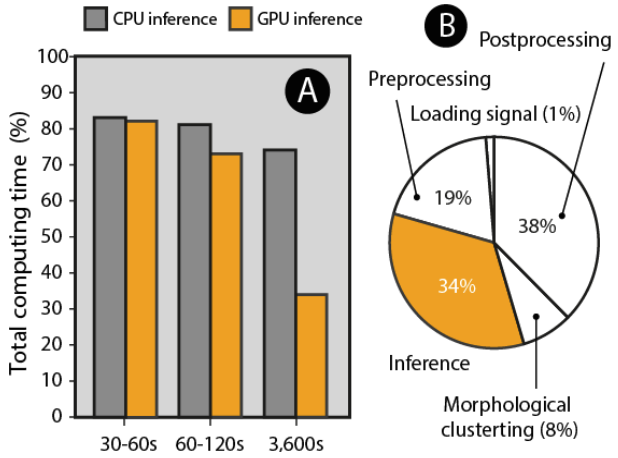

Figure 3. The proportion of inference time in three subsets of benchmark files (A) shows that inference is not the most computationally consuming process when using GPU in long files (3,600 s). The detail of distribution in long files with GPU (B) shows how much other software blocks consume computing time.

5. Discussion

The benchmark test showed AFIB class has the weakest accuracy of all considered rhythm classes, followed by AVBII and NOISE. This can be caused by the fact that deep-learning inference for considering these episodes works in 30 s window (1-second overlap). Consequently, results for AFIB/AVBII and NOISE episodes can start/end only at 30-second intervals. This leads to unprecise starts/ends in cases where, for example, an AFIB episode starts at the 15th second and ends in the 45th second of a recording. On the other hand, VT or SVT episodes depend on a QRS detector model's findings, which might explain their better accuracy. A possible solution to this issue is finding an approach that could point to parts of signals necessary for a specific decision. We experimented with an attention mechanism inspired by [8]; however, we did not reach acceptable results.

Although the computational power in the current deployment in TMC shows enough reserve for computing data in real-time, we would like to decrease the computational complexity further. The first step could be consistent pre-processing for both deep-learning models. Moreover, both deep learning models could be combined into one model with multiple output heads, decreasing the time required for transferring data to the GPU and back. Also, pre-processing (namely FFT band-pass filtering) could be provided by GPU, as we already implemented before [9].

6. Conclusion

We presented scalable and multiplatform ECG processing back-end software for use in the ecosystem of TMC. Scalability means that the number of software instances waiting for data in a shared folder is variable and may be changed on the fly. Therefore, its computing capacity may be adopted to TMC needs, including optional GPU acceleration. In terms of multiplatform deployment, the software currently runs on Windows (physical machine with GPU) and Linux (virtual machines) operating systems.

Although the software is not freely accessible, we are open to considering processing ECG data for research and non-commercial purposes.

Acknowledgments

The research was supported by the Czech Technological Agency grant number FW01010305 and the project RVO:68081731 by the Czech Academy of Sciences.

References

[1] A. Ivora *et al.*, "QRS detection and classification in Holter ECG data in one inference step," *Sci. Reports* |, vol. 12, p. 12641, 123AD.

[2] G. D. Clifford *et al.*, "False alarm reduction in critical care," *Physiol. Meas.*, vol. 37, no. 8, pp. E5--E23, 2016.

[3] F. Plesinger, J. Jurco, J. Halamek, and P. Jurak, "SignalPlant: An open signal processing software platform," *Physiol. Meas.*, vol. 37, no. 7, 2016.

[4] W. McKinney, "Data Structures for Statistical Computing in Python," *Proc. 9th Python Sci. Conf.*, pp. 56–61, 2010.

[5] C. R. Harris *et al.*, "Array programming with NumPy," *Nat. 2020 5857825*, vol. 585, no. 7825, pp. 357–362, Sep. 2020.

[6] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019, vol. 32.

[7] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, 2011.

[8] I. M. Baltruschat, H. Nickisch, M. Grass, T. Knopp, and A. Saalbach, "Comparison of Deep Learning Approaches for Multi-Label Chest X-Ray Classification," *Sci. Reports 2019 91*, vol. 9, no. 1, pp. 1–10, Apr. 2019.

[9] P. Nejedly, F. Plesinger, J. Halamek, and P. Jurak, "CudaFilters: A SignalPlant library for GPU-accelerated FFT and FIR filtering," *Softw. - Pract. Exp.*, 2017.

Address for correspondence:
fplesinger@isibrno.cz