Leo Buron*, Andreas Erbslöh*, Zia Ur-Rehman, Christian Klaes, Karsten Seidl, and Gregor Schiele

Deep.Neural.Signal.Pre-Processor - Towards Development of AI-enhanced End-to-End BCIs

https://doi.org/10.1515/cdbme-2023-1118

Abstract: This paper presents a software-based Python framework for developing future AI-enhanced end-to-end Brain-Computer-Interfaces (BCI). This framework contains modules from the emulated analogue front-end and from neural signal pre-processing for invasive neural applications. These modules can be assembled into several pipeline versions for evaluation and benchmarking. The aim of this framework is to accelerate the development of BCIs due to system-wide optimizations in order to set the requirements for hardware development without prior knowledge on the basis of accuracy (recall and precision) and latency. In the next step, the pipeline can be optimised for on-chip and embedded execution.

Keywords: neural signal processing, extracellular recordings, artificial intelligence, embedded computing

1 Motivation

Diseases and damage to the central nervous system lead to social interaction restrictions and reduced quality of life for those who suffer from them. For example, patients with paraplegia suffer from a partial to complete loss of mobility. Brain-Computer Interface (BCI) research has shown that EEG recordings allow reliable detection of leg and arm movement ambitions that can be used to control an exoskeleton or hand prosthesis [1]. However, EEG recordings capture only low spatial resolution of neural activity at the motor cortex. Invasive microelectrode arrays (MEA), like the Utah array, increase the spatial resolution by capturing fine movements like hand and finger [2]. For real-time determination of the patient's movement ambition, neural signal processing (NSP) of the electrode signals is necessary. Figure 1 shows the example of an end-to-end BCI pipeline for paralyzed patients. This includes (i) the MEA and tissue as signal source, (ii) the analogue processing as front-end, (iii) the neural pre-processor, (iv) the neural decoder and (v) the prosthesis to perform the predicted movement from the neural input. Current experimental platforms digitize the neural input on-chip and transmit the raw data to a workstation telemetrically with data rates in the upper GB/h. There, the NSP with spike sorting and decoding takes place on-/offline with high computational effort. In order to enable calculations close to the patient, a transfer of algorithms from processor to embedded and on-chip solutions must take place in future. In this process, a reduction of the computational complexity and memory space must be achieved with simultaneously high energy efficiency, low latency, high robustness and high accuracy.

This paper presents the Python framework *Deep.Neural.Signal.Pre-Processor* (DeNSPP) to build and evaluate different NSP pipelines for future end-to-end BCIs with AI support. It allows configurable high-level modelling of each pipeline segment on the overall system performance based on defined metrics (see Figure 1). The goals of this framework are (a) to accelerate the development of digital neural pre-processors on embedded devices with an optimized analogue front-end and (b) to explore novel concepts for on-chip and embedded execution.

In the following, the environment of this framework is presented by using AI-based spike sorting for neural signal pre-processing in soft- and future-suitable hardware implementation in neural devices. In general, spike sorting is used to (i) detect spikes from the data stream and (ii) create a spike train that can divide the detected action potentials (spikes) into different clusters based on the signal shape [3] (see example in Figure 1). These clusters can be assigned to a neuron type in the neural decoder, which is used with the recorded local field potential to predict a movement.

2 Framework structure

This section describes the structure for building pipelines within this Python framework DeNSPP. This framework will be made available to the research community in the future targeting (i) software developers for validating their AI algo-

***Corresponding author: Leo Buron,** University of Duisburg-Essen, Faculty of Engineering, Duisburg, Germany, e-mail: leo.buron@uni-due.de
***Corresponding author: Andreas Erbslöh,** University of Duisburg-Essen, Faculty of Engineering, Duisburg, Germany, e-mail: andreas.erbsloeh@uni-due.de
Karsten Seidl, Gregor Schiele, University of Duisburg-Essen, Faculty of Engineering, Duisburg, Germany
Karsten Seidl, Fraunhofer Institute of Microelectronic Circuit and Systems, Business Unit Health, Duisburg, Germany
Zia Ur-Rehman, Christian Klaes, Ruhruniversity Bochum, Faculty of Medicine, Bochum, Germany

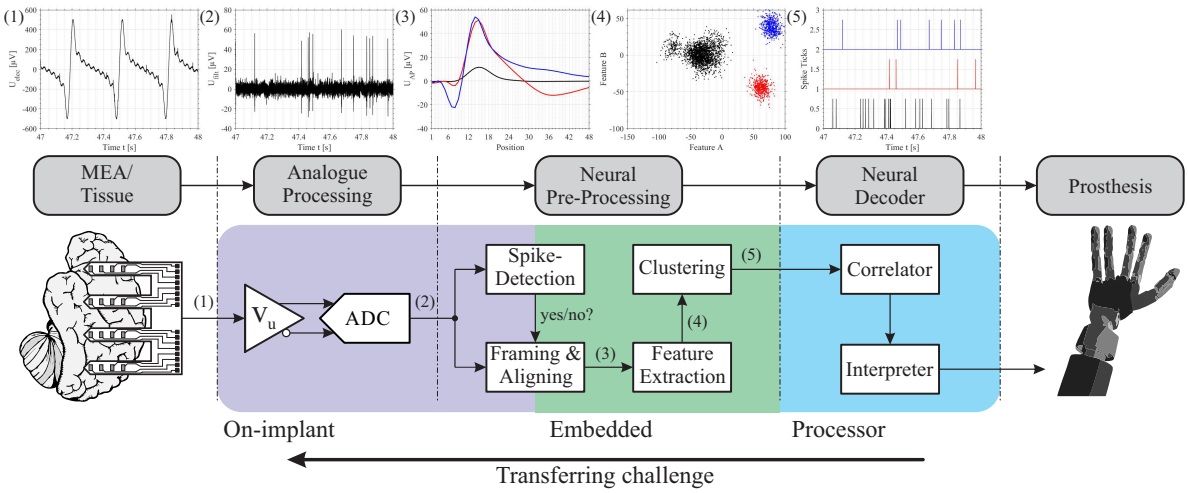

Fig. 1: Neural signal processing pipeline of an end-to-end BCI in order to predict movement from recorded electrode signals of the motor cortex for paralyzed patients including the mapping of the module implementation on different technology platforms

rithms in the decoding and pre-processing stages, (ii) hardware developers for optimizing their system design, and (iii) experimental users for evaluating their electrophysiological datasets. The main selling points of this framework are:

- Standardized module setup for assembling the pipeline
- Emulating the non-idealities from analogue front-end
- Benchmarking different pipelines for the same task
- Reducing the number of hardware iterations
- Hardware-software co-design of AI accelerators
- Evaluating the impact of each module on the pipeline
- Investigating robustness by including artefacts and noise

2.1 Pipeline Setup

The desired pipeline can be built by using pre-defined and standardized modules. Figure 2 shows the configuration of a possible spike sorting pipeline. This includes a (a) data loader (e.g. by using the Quiroga dataset [4]), (b) pre-amplifier, (c) analogue-digital converter (ADC), (d) digital signal pre-processing, (e) spike detection, (f) feature extraction, and (g) clustering solutions.

The data loader prepares the used dataset into a unique data format within this framework. It converts the bitstream of raw data from digital values into electrode voltages and processes the labelling of spike positions and corresponding clusters for supervised and unsupervised learning. In addition, the data can be resampled and windowed. For the analogue pre-amplifiers, the electrical methods (e.g. capacitive coupling, chopping) with their configurations incl. gain, filter characteristic, noise characteristic and electrical non-idealities (e.g. output offset) can be set. The ADCs' (e.g. SAR-, noise shaping SAR-, $\Delta\Sigma$-ADC) characterics (e.g. incl. sampling rate, bit-width, and electrical non-idealities like quantization error)

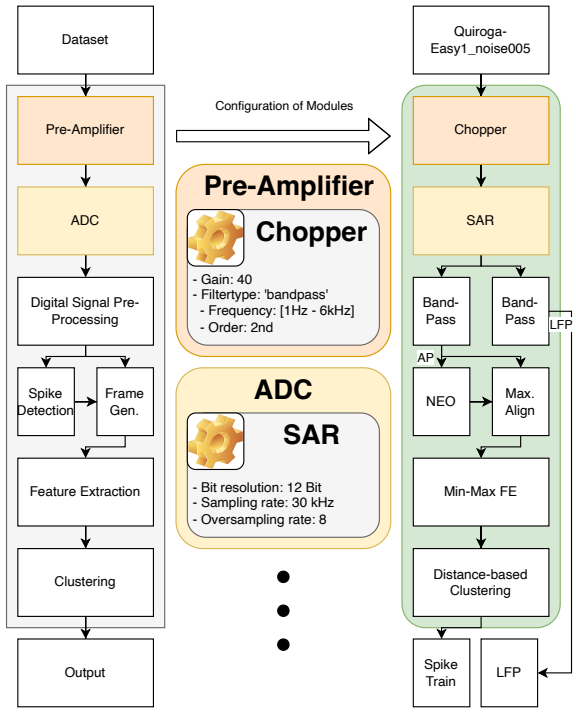

Fig. 2: Configuration example of a linear pipeline: (left) module name - (middle) module properties - (right) used function

are configurable. In digital pre-processing, spatial filters like Common Average Referencing (CAR) for suppressing artefacts and noise, transient filters like bandpass filters for separating spikes, and local field potentials from the pre-amplified signal are provided. In addition, a time delay can be applied to all filters. At this point, the neural pre-processing stage with spike sorting starts with the spike detection on the filtered spike activity. There, different detection algorithms with

optional alignment are offered for the frame generation. The spike frames' features can be extracted in the next phase. There, simple geometric, variance-based (e.g. principle component analysis, PCA), as well as deep neural network-based feature extractors are provided. Either the extracted features or the spike frames themselves can be clustered in the final phase.

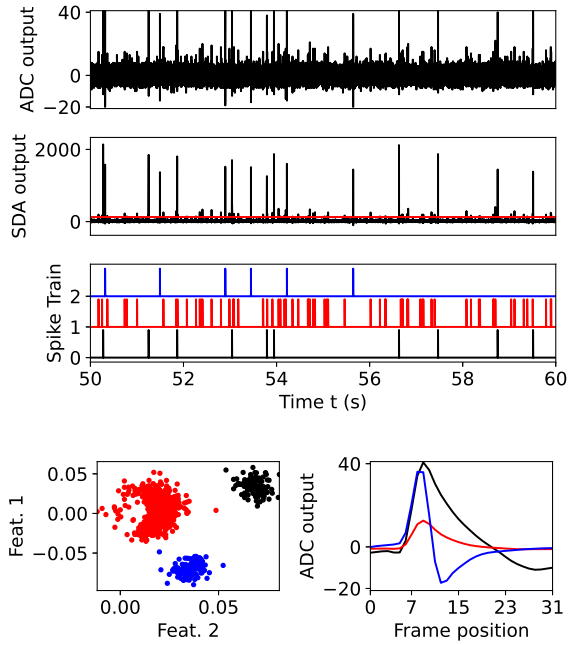

Fig. 3: Example Evaluation of the pipeline

After running the pipeline, tools for the evaluation are provided like in figure 3. It shows the signal output of the ADC, the spike detection algorithms (SDA) output with its threshold and the outgoing spike train. In the lower graphs, a 2D-feature space of a PCA and the corresponding mean waveforms of the clusters are presented. In addition, the framework offers a detailed replay of the signals, feature spaces and clusters. Because of the high-level view of the detailed configuration, overlaying grid searches can be implemented easily to optimize the whole BCIs' or neural implants' performance.

2.2 Using Machine Learning

The pipeline also includes a machine-learning wrapper for a custom training loop. In figure 4 we show an example of a trainings-wrapper for an autoencoder. The wrappers' configuration includes the end of the training, the start of the first inference that is passed to the next module, and the deep learning (DL) specific configuration like the epochs, batch size, optimizer, and loss function. A description of implementing an autoencoder for denoising in the application of spike sorting is given in the next section. Due to the modularization, the trainings-wrapper can be customized for other machine learning frameworks.

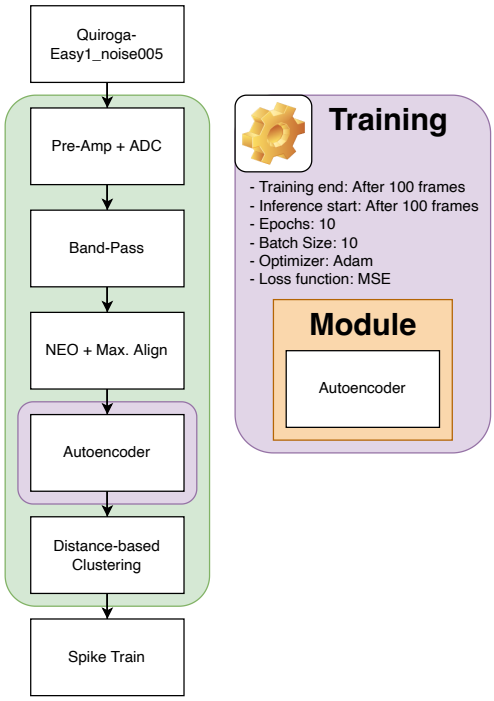

Fig. 4: Configuration example for using an autoencoder in the digital domain for feature extraction

The implementation of deep neural networks on embedded devices requires resource-reducing techniques like quantization. In addition, the implementation on FPGAs introduces hardware artefacts. Addressing this issue, we use our existing open-source ElasticAi-creator tool [5]. It models the introduced hardware artefacts and allows a machine-learning model designer to train the best model for the FPGA without any prior knowledge about it. Today, our solutions model the hardware artefacts in software. As soon as a matching hardware-accelerator design is implemented, the software component will be integrated into the ElasticAi.creator tool allowing a wider audience to benefit from the software-hardware co-design.

3 Implementing a Denoising Autoencoder

This section presents one example of implementing a denoising autoencoder (DAE) for AI-enhanced spike sorting. This DAE performs several tasks at the same time, allowing data reduction between an ASIC and an embedded device, denoising of input spike frames and feature extraction.

Figure 5 shows in (a) a hardware concept of an end-to-end spike sorter with an autoencoder for denoising the detected spike frames with the corresponding signals in (b). In this pipeline, the ASIC includes the analogue front-end (AFE) with digitization and spike detection. The detected spike frames are loaded into the encoder of the autoencoder in order to extract the features which are transmitted to the FPGA. There, these features are loaded into the decoder in order to reconstruct the input spike frames from the encoded features for each recording channel N. These denoised frames can be used for feature extraction and clustering.

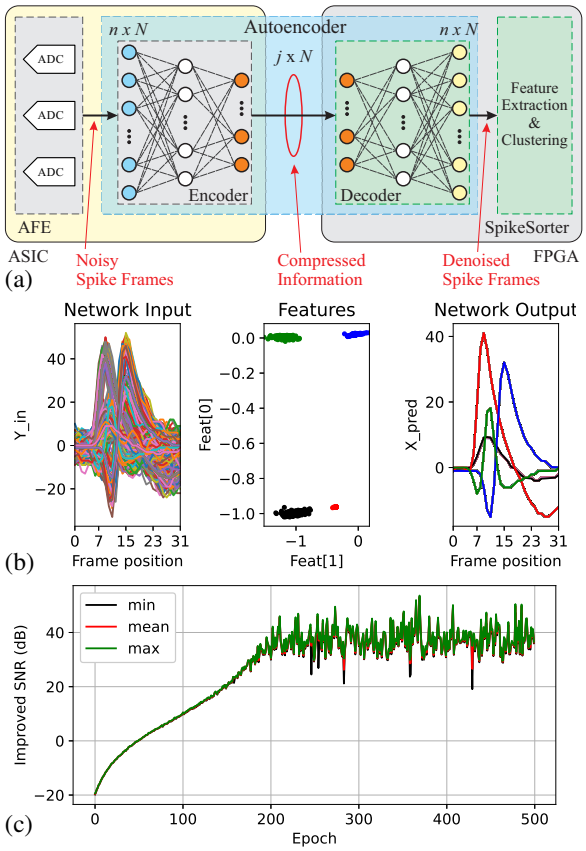

Fig. 5: Concept of an AI-enhanced spike sorting with denoising autoencoder: (a) Hardware pipeline for on-chip and embedded inference - (b) Corresponding signal plots - (c) Improved SNR

We merged the Quiroga data set [4] to four clusters with 5,200 spikes and augmented 2,800 artificial noisy spikes from the merged data set. Then we processed the data with our emulated front-end like in Figure 5(a). Prior to the first linear layer, batch normalization is performed. The DAE has five linear layers with tanh as activation function except for the last layer. The five 1d-linear layers have 32-20-3-20-32 perceptrons. For supervised training in software, we split the dataset to 0.8/0.2 test/validation and use the Adam optimizer with MSE loss at a batch size of 256. The ElasticAI.creator tool is used for quantised training [5] at 12-bit resolution with 9-bit fraction of all layers.

The results of the training are shown in Figure 5(b,c). This DAE increases the signal-to-noise ratio (SNR) of the noisy input spike frames in the range of $[-3.18\,\mathrm{dB},\ 2.81\,\mathrm{dB}]$ to $[37.48\,\mathrm{dB},\ 42.89\,\mathrm{dB}]$. On average, an SNR improvement of $41.14\,\mathrm{dB}$ is achieved. In addition, this setup reduces the data rate per electrode channel from transferring the raw data with $240\,\mathrm{kbit/s}$ (sampling rate: $20\,\mathrm{kHz}$, quantization: 12-bit) to $1.688\,\mathrm{kbit/s}$ (features of hidden layer with window length $1.5\,\mathrm{ms}$ and 50 spikes/s). Compared to transmitting only the spike frames after spike detection, a data reduction of factor 142 is achieved.

4 Conclusion

The Deep.Neural.Signal.Pre-Processor Python framework allows developing and benchmarking signal processing pipelines in software for different use cases for time series analyses of electrophysiological recordings. Specific to this package is that the analogue-front end is emulated in software, allowing non-experts to find out their requirements for the hardware implementation. In addition, different AI models can be evaluated and optimized for embedded running with high resource-, energy-efficiency, low-latency approaches and high robustness against artificial artefacts and noise.

In future, more modules for signal processing and for neural decoding will be included in this framework in order to enable research on new system topologies and algorithms e.g. neural implants for closed-loop stimulation and BCIs for movement prediction.

References

[1] Al-Quraishi M, Elamvazuthi I, Daud SA, Parasuraman S, Borboni A. EEG-Based Control for Upper and Lower Limb Exoskeletons and Prostheses: A Systematic Review. Sensors 2018;18:10. DOI: 10.3390/s18103342 .

[2] Aflalo T, Kellis S, Klaes C, Lee B, Shi Y, Pejsa K et al. Decoding motor imagery from the posterior parietal cortex of a tetraplegic human. Science 2015;348:906–910. DOI: 10.1126/science. aaa5417.

[3] Gibson S, Judy JW, Markovi DC. Spike Sorting: The First Step in Decoding the Brain. IEEE Signal Processing Magazine 2012;29:124–143. DOI: 10.1109/MSP.2011.941880.

[4] Quiroga RQ. Simulated dataset [Online]. Feb. 2020. Available: https://figshare.le.ac.uk/articles/dataset/Simulated_dataset/11897595.

[5] Qian C, Einhaus L, Schiele G. ElasticAI-Creator: Optimizing Neural Networks for Time-Series-Analysis for on-Device Machine Learning in IoT Systems. SenSys '22 2023;;941–946. DOI: 10.1145/3560905.3568296.