Spoken Language Understanding (SLU) systems automate the process of extracting the meanings or semantics of human speech. Conventional SLUs employ two principal blocks: an Automatic Speech Recognition (ASR) which transcribes spoken words into text and Natural Language Understanding (NLU) which analyzes the intents (Figure 1, left).

It is worth mentioning that these unified solutions also simplify the training-aware or post-training preparation processes necessary for quantized inferences in the embedded devices. Subsequently, in this study, a novel end-to-end solution is introduced using recurrent neural networks is proposed. The proposed unified network structure is trained and evaluated on a publicly available dataset, and execution reports on the RISC-V based GAP and GAP9 platforms are described.

**Methods: modeling assumptions**

The principal idea is to predict a tuple of intents/slots from a sequence of acoustic features calculated from the spoken utterance. This can be achieved by solving a classification problem using conventional techniques; however, it is assumed that a dataflow stream from the input sequence (i.e., acoustic features) to the first slot and so on till reaching the last slot. Subsequently, an output sequence can simply be formed by a typical order of slots. Also, in order to indicate the starting and ending points of the output sequence, two additional slots (±) with certain values are added accordingly:

\[
\text{Output sequence } = \{ \text{START}, S_0, S_1, \ldots, S_m, S_n, \text{END} \} \tag{1}
\]

This helps to differentiate the beginning and ending of the predicted output sequence from the intermediate sections.

**Methods: model design**

A very detailed architecture of the proposed seq-to-seq modeling is illustrated in Figure 3. Both Encoder and Decoder modules are implemented using recurrent layers. The encoder layer is fed with acoustic features to generate a high level of abstraction from the spoken utterance. Once the encoder processes all input sequence, the extracted features are transferred to the Decoder module by initialising of its recurrent layer. As can be seen, the Decoder is followed by a fully connected layer to map the extracted embedding into the target vocabulary of utterances. As mentioned earlier, the output sequence is supposed to start and finish with additional slots of ‘START’ and ‘END’ respectively. This means that the Decoder is first fed by ‘START’ in order to predict the rest of the slots until it reaches the ‘END’ slot.

**Methods: Quantization**

For the quantization and deployment of the SLU algorithm, we used NNTool, our internally developed neural network compressor for GAP processors. NNTool is a python library that aims at simplifying a high level DSP/NN graph description (provided as an onnx or tflite model) by converting the computational nodes into GAP operators, i.e. SW kernels implemented in our backend library, the Autotiler[2].

Besides standard toplogy optimizations, such as operation fusions and offline tensors reorganization for more efficient computation, NNTool enables the automatic quantization of different neural networks and DSP layers. Different types of quantization schemes are available in the backend of NNTool, including 16 and 8bits Scaled and POW2 fixed point and IEEE16 floating point quantization. In this poster, we will focus on the Scaled 8 bits quantization employed in the SLU application. Following [3], our Scaled quantization maps every quantized tensor of the NN inference to the real numerical space with the affine transformation:

\[
r = \frac{s - \sigma}{\sigma'} \tag{3}
\]

where \( q \) is the integer value quantized to 8bits, \( r \) is the representation of that value in the real space, \( s = M \cdot 2^{\delta} \) the scaling factor, represented in our tools and SW library with 8bits for the mantissa \( \delta \) and 16bits for the exponent \( N \), and \( Z \) the Zero point, the integer representation of the 0, also in 8bits as the value \( q \). The LSTM is the key operator of SLU, it uses its previous output (state) along with the new data (input frame) to recursively extract the command embedding without losing the long term memory of the sentence.