Python Wrapper for Context-based Adaptive Binary Arithmetic Coding

Christian Rohlfing, Thibaut Meyer, Jens Schneider
Institut für Nachrichtentechnik,
RWTH Aachen University, Germany
{rohlfing, thibaut.meyer, schneider}@ient.rwth-aachen.de

Jan Voges
Institut für Informationsverarbeitung,
Leibniz University Hannover, Germany
voges@tnt.uni-hannover.de

***Abstract*—A Python wrapper for Context-based Adaptive Binary Arithmetic Coding (CABAC), extracted from the Test Model (VTM) for Versatile Video Coding (VVC), is presented. Besides providing Python access to CABAC, two extensions are proposed: the probability estimation progress for each context can be traced, and sequences of integer values can be coded without designing a dedicated context model set.**

I. Introduction

The entropy coding method Context-based Adaptive Binary Arithmetic Coding (CABAC) [1] was introduced in the video compression standard Advanced Video Coding (AVC) and is still used in its successors High Efficiency Video Coding (HEVC) and Versatile Video Coding (VVC) for entropy coding of all syntax elements. Note that a more efficient core coding engine was introduced in VVC [2].

The CABAC coding procedure can be summarized as follows: integer symbols are binarized in a first step, resulting in a sequence of binary values (bin-string). Each bin is then passed to the arithmetic coding engine, operating with two modes: the faster bypass mode codes bins using a fixed probability of $p = \frac{1}{2}$ whereas the more complex regular mode estimates the probability using adaptive models, i.e. contexts, which are chosen based on previously coded information. We refer to [2] for a detailed explanation of the VVC variant of CABAC.

This paper introduces PyCABAC, a Python wrapper for CABAC. The core part of CABAC was extracted from the VVC reference software of VVC (VVC Test Model, VTM). The corresponding C++ code was then exposed to Python via pybind11 [3] bindings, resulting in a Python package which can be installed via the package installer `pip`. Two extensions to the core part were implemented: firstly, the current state of the probability estimator after the corresponding update step can be traced, enabling the evaluation of the efficiency of each context. Secondly, symbol sequences can be coded combining binarization, the corresponding context selection and the subsequent bin-wise coding, providing a beginner-friendly entry point to understanding CABAC. The code of PyCABAC is available online[1].

II. Implementation

Four Python/C++ classes for encoding (and three corresponding classes for decoding) are available:

[1]https://github.com/ient/pycabac

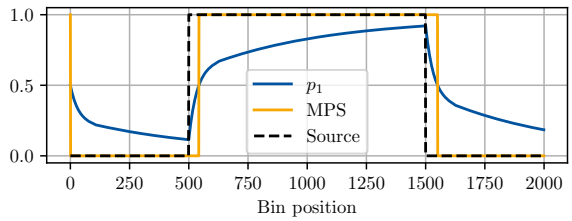

Fig. 1: Tracing of p_1 and MPS for a simple binary source.

1) `cabacEncoder` provides the core CABAC part including context-dependent or bypass encoding of bins.
2) `cabacTraceEncoder` keeps track of the value of the most probable symbol (MPS) as well as p_1, the currently modelled probability of the symbol being equal to 1 after each context update step as shown in Fig. 1.
3) `cabacSymbolEncoder` accepts integer values as input which are binarized and then encoded bin-wise. For context-coding, a function providing the corresponding context ID for each bin has to be provided as well. The following binarization schemes are supported: Direct binary representation of N bits, truncated unary or exponential-Golomb coding of order k.
4) `cabacSimpleSequenceEncoder` accepts a sequence of integer values and provides some simplistic context model sets for context-dependent coding. For example, a unique context is assigned to each combination of the position of the to-be-coded bin in the bin-string and the values of previously coded symbols.

For a beginner-friendly access, several demo and test cases were developed as well which are also available online.

References

[1] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 620–636, Jul. 2003.

[2] H. Schwarz, M. Coban, M. Karczewicz, T.-D. Chuang, F. Bossen, A. Alshin, J. Lainema, C. R. Helmrich, and T. Wiegand, "Quantization and entropy coding in the versatile video coding (vvc) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 10, pp. 3891–3906, 2021.

[3] W. Jakob, J. Rhinelander, and D. Moldovan, "pybind11 — seamless operability between c++11 and python," 2016. [Online]. Available: https://github.com/pybind/pybind11