Architecture-Aware Neural Compression for TriCore Firmware using Knowledge Distillation

Hyunjoong Lee, Hoseong Kim and Daejin Park
School of Electronic and Electrical Engineering
Kyungpook National University Daegu, Republic of Korea

***Abstract*—The transition toward Software-Defined Vehicles (SDVs) has significantly increased the complexity and size of automotive ECU firmware, posing growing challenges in hardware cost (BOM) and over-the-air (OTA) update efficiency. Conventional general-purpose compressors fail to exploit the architectural characteristics of the TriCore processor, resulting in limited compression efficiency. To address this limitation, this paper proposes an AI-driven, architecture-aware compression framework tailored for TriCore-based firmware. The proposed method employs a two-stage knowledge distillation scheme, in which a large Transformer-based Teacher model transfers its learned representations of TriCore binary patterns to a lightweight GRU-based Student model suitable for on-device deployment in real ECUs. Experimental results show that the Student-based compression framework achieves an average compression efficiency of 1.524 bpb, corresponding to 19.05% of the original firmware size on the `TC375` dataset, which represents an improvement of approximately 38% compared to conventional general-purpose compressors. In addition, the proposed approach exhibits a compact runtime memory footprint during decompression, requiring only 0.25 MB, which is approximately 94% lower than that of traditional methods. Furthermore, cross-validation between the Python reference implementation and the C-based decoder confirms deterministic and consistent decoding behavior, demonstrating the practicality and portability of the proposed framework. Overall, this study establishes a strong proof of concept for AI-driven, architecture-aware firmware compression and highlights its potential for further optimization and deployment on embedded hardware.**

***Keywords*—TriCore, Firmware Compression, Lightweight AI, Knowledge Distillation**

I. Introduction

Modern automotive systems are undergoing a rapid transition toward the Software-Defined Vehicle (SDV) paradigm, leading to an unprecedented escalation in in-vehicle software complexity. Unlike early generations of Electronic Control Units (ECUs) that primarily executed simple control functions, contemporary vehicles integrate a wide range of high-level software components, including Advanced Driver Assistance Systems (ADAS), artificial intelligence algorithms for autonomous driving, and powertrain electrification modules [1]. This increasing software complexity inevitably expands firmware codebases, resulting in the rapid growth of ECU-deployed binary sizes.

This study was supported by the BK21 FOUR project (4199990113966), the Basic Science Research Program (RS-2018-NR031059, 10%), (RS-2025-24322979, 10%) through the National Research Foundation of Korea (NRF) funded by the Ministry of Education. This work was partly supported by an Institute of Information and communications Technology Planning and Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2022-0-01170, 20%) and (No. RS-2023-00228970, 30%) and (No. RS-2025-02218227, 20%) and (No. RS-2022-00156389, Innovative Human Resource Development for Local Intellectualization support program, 10%). The EDA tool was supported by the IC Design Education Center (IDEC), Korea.

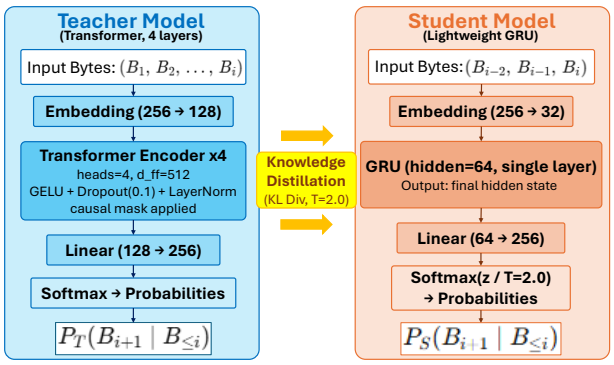

Fig. 1. System Model

The rapid expansion of firmware introduces two critical bottlenecks in both hardware and operational domains. First, the growing demand for larger flash memory capacity directly increases hardware costs. Second, during Over-the-Air (OTA) updates, larger binary sizes lead to higher network bandwidth consumption and longer transmission times, ultimately degrading overall system reliability.

To mitigate these challenges, conventional general-purpose lossless compression algorithms, such as zlib, bzip2, and LZMA, have been widely adopted. However, these algorithms exhibit fundamental limitations in modeling the intrinsic structural patterns of machine code. For example, LZ-based methods can identify simple byte-level repetitions but fail to capture semantic redundancies among instructions that differ only in register allocation. Similarly, statistical models such as bzip2 rely primarily on short-range contexts, making them ineffective for modeling long-range dependencies, including function calls and branch instructions, that frequently appear in compiled firmware binaries.

To address these limitations, this study introduces an AI-driven, architecture-aware entropy modeling framework designed to learn the structural regularities inherent to the TriCore architecture. Neural entropy modeling has been extensively explored in learned compression literature, where neural networks are used to estimate conditional probability distributions for entropy coding [2]. The proposed approach

employs a Transformer-based model that conceptualizes TriCore firmware binaries as a form of "language," learning the conditional probability distribution for next-byte prediction.

However, deploying a large-scale Transformer model directly on an ECU is impractical due to its limited computational and memory resources [3]. To overcome this constraint, this study proposes a two-stage knowledge distillation framework based on the Teacher–Student paradigm. In the first stage, a Transformer-based Teacher model learns complex structural and contextual patterns from TriCore binaries. In the second stage, a lightweight GRU-based Student model distills the Teacher's predictive behavior, achieving comparable functionality with substantially lower computational overhead.

Experimental results demonstrate that the proposed Student model not only outperforms general-purpose compression algorithms in terms of compression ratio but also enables deterministic and memory-efficient decompression, making it practically deployable in real ECU environments.

II. Proposed Architecture

As discussed in the Introduction, conventional general-purpose compressors such as zlib and LZMA exhibit inherent limitations in exploiting the complex structural and semantic regularities embedded in TriCore binaries, resulting in sub-optimal compression efficiency. To address these limitations, this study adopts an AI-driven entropy modeling approach. Specifically, the compression process is reformulated as a next-symbol probability prediction problem, effectively casting it as a probabilistic language modeling task. Under this formulation, TriCore firmware binaries are treated as a form of "language," enabling the model to learn causal dependencies and contextual relationships among byte sequences.

To this end, this study introduces a Transformer-based Teacher model. The Transformer architecture, equipped with a self-attention mechanism, is well suited for capturing long-range contextual dependencies [4], which are essential for modeling the structural characteristics of TriCore code, including branch behaviors, register interactions, and function-call patterns. The Teacher model processes input sequences of 2048 bytes and is trained to predict the conditional probability distribution of the next byte, denoted as $P(x_t \mid x_{<t})$. During training, the Teacher model minimizes the cross-entropy loss, which serves as an empirical estimate of the information entropy $H(x)$ of the training dataset. Through this optimization process, the model learns the statistical regularities inherent in TriCore binaries and produces a reference probability distribution that forms the foundation of the proposed compression framework.

To bridge the gap between the Teacher model's predictive capability and the computational constraints of embedded environments, this study proposes a two-stage knowledge distillation framework [5]. In the first stage, the Transformer-based Teacher model is trained to learn the probabilistic structure of TriCore binaries and to generate conditional byte-level probability distributions, $P_T(B_{i+1} \mid B_{\leq i})$, which serve as reference targets for distillation.

In the second stage, the probability distributions generated by the Teacher model are used as soft targets to guide the training of the lightweight Student model, which is explicitly designed to operate within the limited Flash and RAM resources of automotive ECUs. The Student model is trained to approximate the Teacher's predictive distribution by minimizing a distillation loss, where $P_T(\cdot)$ and $P_S(\cdot)$ denote the output distributions of the Teacher and Student models, respectively.

The Student model employs a single-layer Gated Recurrent Unit (GRU) architecture, replacing the computationally intensive attention mechanism with a short contextual input consisting of only three consecutive bytes, denoted as (B_{i-2}, B_{i-1}, B_i). As a lightweight recurrent neural network optimized for sequential data, the GRU offers two key advantages: a compact memory footprint and high portability when implemented in C. As a result, the Student model internalizes the information-theoretic knowledge distilled from the Teacher while maintaining computational efficiency suitable for embedded decompression on TriCore ECUs. The overall architectures of the Teacher and Student models are illustrated in Fig. 1.

The Teacher and Student models are deliberately designed with contrasting architectural characteristics to balance modeling expressiveness and embedded feasibility. The Teacher model leverages a self-attention mechanism to capture long-range dependencies across instruction sequences, enabling effective modeling of control-flow structures and function-level patterns in TriCore firmware. This global receptive field is essential for learning high-quality probabilistic representations, but comes at the cost of substantial computational and memory requirements.

In contrast, the Student model prioritizes deployment efficiency by replacing attention with a compact recurrent structure and operating on a short local context. While this design inherently limits the accessible temporal range, the Student successfully retains the core predictive behavior of the Teacher through knowledge distillation. This architectural simplification enables practical on-device decompression on TriCore-based ECUs while maintaining competitive compression performance.

The Student model is built upon a GRU-based architecture to enable efficient execution in resource-constrained embedded environments. Compared to Transformer-based models, GRUs process sequential data with substantially fewer parameters and computational operations, making them well suited for on-device deployment [6]. The Student model receives a short contextual input of three consecutive bytes, (B_{i-2}, B_{i-1}, B_i), and is trained via knowledge distillation to approximate the predictive behavior of the Teacher model. This lightweight design preserves the essential statistical characteristics learned by the Teacher while ensuring compatibility with the limited Flash and RAM resources of TriCore ECUs.

Fig. 2 summarizes the training behavior of the proposed two-stage framework. Fig. 2(a)–(b) show that the Teacher model converges stably, with smoothly decreasing training and validation bpb and well-behaved optimization dynamics

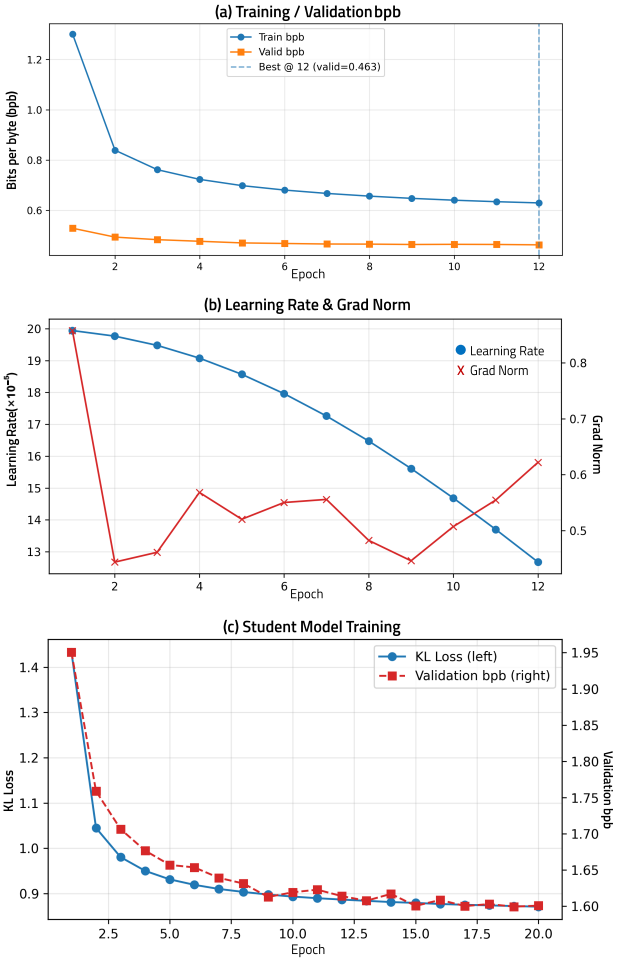

Fig. 2. Training Results of Teacher-Student

under a cosine learning rate schedule. The resulting probability estimates are well calibrated and serve as reliable reference distributions for knowledge distillation. Fig. 2(c) presents the distillation results for the Student model. The KL divergence between the Student and Teacher distributions decreases steadily, while the validation bpb improves and converges to approximately 1.6. Although the Student does not match the Teacher's absolute predictive performance, it achieves an effective trade-off between accuracy and computational efficiency with a substantially smaller context window.

The trained Student model does not operate as a standalone compressor but functions as a probabilistic predictor that estimates the likelihood of the next byte. The predicted probability distributions are integrated with a classical arithmetic coding scheme [7] to perform entropy-based compression and deterministic decoding. During compression, at each timestep i, the most recent three bytes (B_{i-2}, B_{i-1}, B_i) are provided as input to the Student model. The model outputs logits for all 256 possible next-byte symbols, which are converted into a probability distribution using a numerically stable `softmax` function implemented in `float64` precision. These probabilities are then transformed into a 16-bit quantized cumulative distribution function (CDF) by a `probs-to-quantized-cdf` procedure [8], which is directly consumed by the arithmetic encoder.

This process incorporates three implementation-level mechanisms to ensure numerical consistency: minimum frequency enforcement (equivalent to Laplace smoothing with a $+1$ count), rounding behavior consistent with Python's `np.round()` function (Banker's rounding), and a sum-correction procedure applied to the quantized CDF. Together, these mechanisms ensure deterministic behavior between the Python reference implementation and the C-based deployment [9]. Finally, the `RangeEncoder` updates its internal states $(low, high)$ using the generated CDF and the observed symbol, emitting the corresponding bit sequence to the output stream.

The decompression procedure mirrors the compression process and is designed to operate within a TriCore ECU environment. Initially, the `RangeDecoder` reads the compressed bitstream and initializes its internal states $(code, low, high)$. During initialization, the first 32 bits of the bitstream are preloaded to set the *code* variable, following the same initialization strategy as the Python-based `RangeDecoder`. Bit-level alignment between the Python and C implementations was verified to ensure consistent decoding behavior.

At each timestep i, the C-implemented `forward()` function receives the three most recently reconstructed bytes (B_{i-2}, B_{i-1}, B_i) as input and computes the corresponding logits. These logits are processed through a double-precision `softmax` function followed by a `probs-to-cdf` transformation to produce an integer-valued CDF identical to that used during compression. The `RangeDecoder` compares the current bitstream state with this CDF to recover the next symbol B_{i+1} and updates its internal states $(code, low, high)$ accordingly. The decoded symbol is written to the output buffer and used to update the input context for the subsequent prediction step. The detailed structure of this compression–decompression process is illustrated in Fig. 3.

To evaluate the effectiveness of the proposed TriCore architecture-aware compression framework and to enable comparison with conventional general-purpose compressors, a comprehensive experimental setup was established. The evaluation was conducted using approximately 60 compiled firmware binaries generated from official example codes for Infineon's `AURIX TC375` microcontroller. The dataset covers a diverse set of ECU-relevant functionalities, including ADC, UART, PWM, DMA, Timer, and CAN communication modules, representing realistic firmware components commonly used in production vehicles.

For model training, hyperparameter tuning, and evaluation, the dataset was randomly divided into training (70%), validation (15%), and test (15%) subsets. The final compression performance was assessed exclusively on the test set, which was not used during training or validation. A comparative analysis between the proposed model-based compressor and conventional lossless algorithms is presented in Fig. 4, considering compression efficiency (bits-per-byte), runtime memory usage, and throughput.

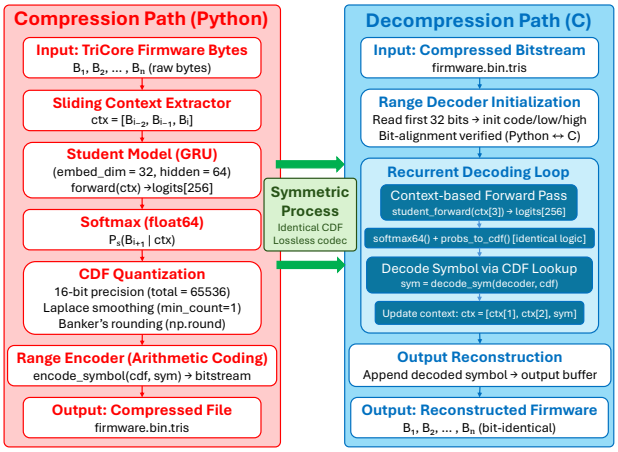

Fig. 3. Compression and Decompression Pipeline

A comprehensive performance comparison between the proposed model-based compressor and conventional general-purpose lossless algorithms is summarized in Fig. 4. The evaluation considers three key metrics: compression efficiency measured in bits per byte (bpb), runtime memory usage during decompression, and decompression throughput. For clarity, the quantitative performance comparison corresponding to Fig. 4 is summarized in Table I.

As shown in Fig. 4(a), the conventional compressors ZLIB, BZIP2, and LZMA achieve compression efficiencies of 2.628 bpb (32.85%), 3.055 bpb (38.19%), and 2.457 bpb (30.71%), respectively. In contrast, the proposed TriCore-aware model achieves 1.524 bpb, corresponding to 19.05% of the original data size. This represents an improvement of approximately 38% in compression efficiency compared to LZMA, which provides the strongest baseline performance among the conventional methods.

Fig. 4(b) shows the runtime memory usage during decompression. The proposed model requires an average of 0.25 MB of memory, representing a reduction of approximately 94% compared to ZLIB, which exhibits the lowest memory consumption among the conventional compressors at 4.25 MB. Similar memory usage levels are observed for BZIP2 (4.27 MB) and LZMA (4.31 MB), highlighting the compact nature of the proposed decoder in memory-constrained ECU environments.

As shown in Fig. 4(c), the proposed model exhibits lower decompression throughput than conventional lossless compressors. This limitation is primarily attributable to the computational overhead of neural network inference, which has not yet been optimized in the current implementation. It should be noted that the primary objective of this study is to evaluate the feasibility and compression benefits of model-based decoding under strict ECU memory constraints, rather than to achieve maximum decompression throughput at this stage.

Despite the remaining trade-off in decompression speed, the results clearly demonstrate that the proposed architecture-aware approach effectively addresses the limitations of

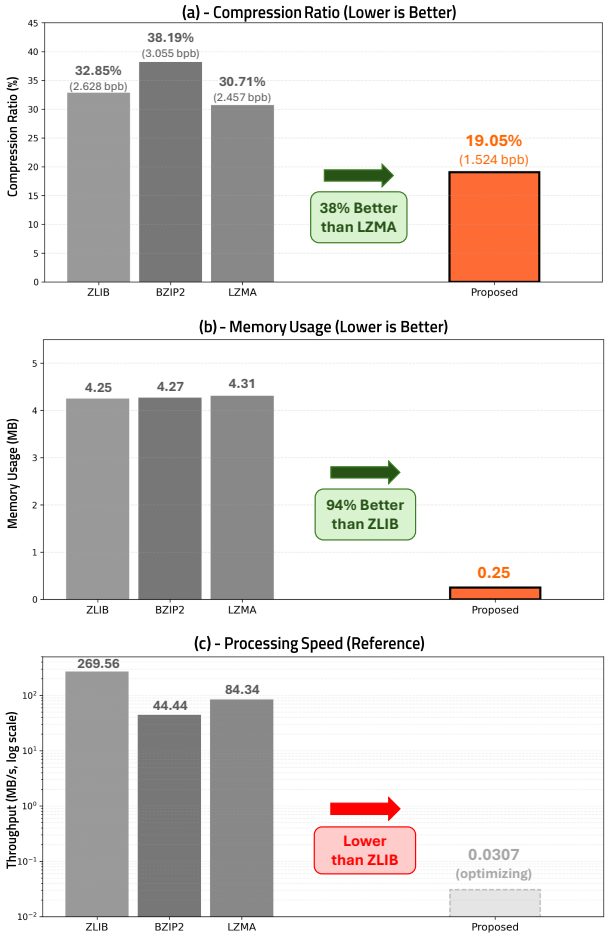

Fig. 4. Compression Performance and Runtime Memory on TriCore Firmware

TABLE I
COMPRESSION PERFORMANCE AND RUNTIME MEMORY

Method	bpb	Runtime Mem. (MB)
ZLIB	2.628	4.25
BZIP2	3.055	4.27
LZMA	2.457	4.31
Proposed (Student + Arithmetic)	**1.524**	**0.25**

architecture-insensitive compressors discussed in the Introduction, particularly in exploiting structural redundancy within TriCore firmware binaries. By learning intrinsic byte-sequence regularities specific to the target architecture, the proposed method achieves superior compression efficiency and substantial runtime memory savings. These findings provide a strong proof of concept and suggest considerable potential for further improvements in decoding throughput through architectural simplification, model quantization, or dedicated hardware acceleration.

Furthermore, to verify the practical deployability of the proposed model on ECUs, a functionally equivalent decompression implementation was developed in C and validated against the Python reference version. The validation was conducted in a PC environment to ensure consistent decoding

behavior across both implementations. The results confirmed that the model's output logits for the initial prediction context were numerically consistent within floating-point precision limits. In addition, the initial 32-bit internal state of the `RangeDecoder` was verified to be bitwise identical, demonstrating deterministic decoding behavior under matched numerical conditions.

Overall, the validation results confirm functional equivalence between the C-based decoder and the Python reference implementation under controlled numerical precision settings. The proposed C implementation therefore constitutes a faithful and deployable realization of the trained Student model. This validation demonstrates that the proposed compression and decompression pipeline achieves high portability and practical feasibility for deployment within real TriCore ECU environments.

III. Conclusion

This paper presented an AI-driven, TriCore architecture-aware compression framework designed to address the critical challenge of rapidly increasing ECU firmware size in the era of Software-Defined Vehicles (SDVs). To enable deployable architecture-aware compression on embedded ECUs, a two-stage Teacher–Student knowledge distillation framework was introduced to transfer the statistical knowledge of a high-capacity Transformer-based Teacher model to a compact GRU-based Student model.

Experimental results showed that the proposed compression framework, which integrates the lightweight Student model with an arithmetic coding scheme, achieves an average compression efficiency of 1.524 bpb, corresponding to approximately 19.05% of the original firmware size on the `TC375` example-code dataset. This represents an improvement of approximately 38% compared to LZMA, which provides the strongest performance among the conventional general-purpose compressors. In addition, the proposed method exhibits substantially reduced runtime memory usage during decompression, requiring an average of 0.25 MB, which is approximately 94% lower than that of ZLIB.

Although the current implementation has not yet been optimized for maximum decompression throughput, the experimental results validate the effectiveness of the proposed architecture-aware design under strict embedded memory constraints. By capturing structural regularities specific to TriCore binaries, the proposed framework achieves substantially improved compression efficiency and decompression-time memory usage compared to conventional architecture-insensitive compressors, while maintaining a lightweight runtime footprint suitable for embedded environments.

The primary contribution of this study lies in the design of an AI-based knowledge distillation pipeline tailored to real-world embedded constraints, as well as the successful PC-level validation of a high-efficiency entropy-model-based compression engine specialized for TriCore firmware. Despite being trained on a relatively limited dataset, the proposed approach demonstrates substantial performance gains over general-purpose compressors, highlighting its potential for broader applicability and scalability.

While this work primarily demonstrates the feasibility of architecture-aware neural compression for TriCore firmware, future research will focus on improving decompression throughput and overall model efficiency for deployment in real ECU systems. The validated C implementation will be ported to an actual TriCore target board to evaluate on-device decompression latency and memory consumption. In addition, future studies will address numerical robustness under embedded compiler environments and expand the dataset with larger-scale TriCore firmware binaries to further enhance performance and generalization.

References

[1] K. Yadav, S. Shinde, C. Varsha, A. Kumar, V. Ramakrishna, and P. Rajalakshmi, "Design considerations and framework analysis for software-defined autonomous vehicles," in *2024 IEEE 99th Vehicular Technology Conference (VTC2024-Spring)*, 2024, pp. 1–5.

[2] A. B. Yeşilyurt and F. Kamışlı, "End-to-end learned image compression with conditional latent space modeling for entropy coding," in *2020 28th European Signal Processing Conference (EUSIPCO)*, 2021, pp. 501–505.

[3] L. Qin and J. Sun, "Model compression for data compression: Neural network based lossless compressor made practical," in *2023 Data Compression Conference (DCC)*, 2023, pp. 52–61.

[4] T. V. Kale and S. Mendhe, "A review on advances in sentiment analysis: A deep learning approach using transformer based models," in *2025 4th International Conference on Sentiment Analysis and Deep Learning (ICSADL)*, 2025, pp. 235–239.

[5] R. Sun and P. Jian, "A two-stage distillation method: Teach model to answer questions after comprehending the document," in *2023 International Conference on Asian Language Processing (IALP)*, 2023, pp. 240–245.

[6] Y. Zhang, "Prediction model of uav damage efficiency based on gru neural network," in *2024 10th International Conference on Mechanical and Electronics Engineering (ICMEE)*, 2024, pp. 334–338.

[7] H. Ye, G. Deng, and J. Devlin, "A lossless image compression system using a binary arithmetic coder," in *ICSP '98. 1998 Fourth International Conference on Signal Processing (Cat. No.98TH8344)*, 1998, pp. 819–822 vol.1.

[8] J. Zhang and D. Berleant, "Envelopes around cumulative distribution functions from interval parameters of standard continuous distributions," in *22nd International Conference of the North American Fuzzy Information Processing Society, NAFIPS 2003*, 2003, pp. 407–412.

[9] R. Yang, D. Liu, F. Wu, and W. Gao, "Learned image compression with efficient cross-platform entropy coding," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 15, no. 1, pp. 72–82, 2025.