All You Need is Feedback: Communication with Block Attention Feedback Codes

Emre Ozfatura, Yulin Shao, Alberto Perotti, Branislav Popovic, Deniz Gündüz

Abstract—Deep neural network (DNN)-based channel code designs have recently gained interest as an alternative to conventional coding schemes, particularly for channels in which existing codes do not provide satisfactory performance. Coding in the presence of feedback is one such problem, for which promising results have recently been obtained by various DNN-based coding architectures. In this paper, we introduce a novel learning-aided feedback code design, dubbed *generalized block attention feedback (GBAF) codes*, that achieves order-of-magnitude improvements in block error rate (BLER) compared to existing solutions. Sequence-to-sequence encoding and block-by-block processing of the message bits are the two important design principles of the GBAF codes, which not only reduce the communication overhead, due to fewer interactions between the transmitter and receiver, but also enable flexible coding rates. GBAF codes also have a modular structure that can be implemented using different neural network architectures. In this work, we employ the popular transformer architecture, which outperforms all the prior DNN-based code designs in terms the BLER in the low signal-to-noise ratio regime when the feedback channel is noiseless.

Index Terms—Feedback code, deep learning, deep neural networks, channel coding, the attention mechanism, self-attention, ultra-reliable short-packet communications, transformer.

I. INTRODUCTION

Reliable communication in the presence of noise has been a long-standing challenge. Numerous coding and modulation techniques have been invented over many decades to push the boundaries of communication; that is, to achieve higher data rates with less error probability under given resource constraints (bandwidth, power). Information storage and communication are two core technologies that underpin the information age, and the success of both hinges on error correction codes, such as BCH, Reed-Muller, convolution, turbo, low-density parity-check (LDPC), and polar codes. While these codes can approach the fundamental Shannon capacity limit over an additive white Gaussian noise (AWGN) channel in the large block length regime, there are many scenarios where we do not have practical codes that approach the fundamental theoretical boundaries.

Coding in the presence of feedback is one such challenging, yet practical scenario. The classical feedback channel model

E. Ozfatura, Y. Shao and D. Gündüz are with Information Processing and Communications Lab, Department of Electrical and Electronic Engineering, Imperial College London. Emails: {m.ozfatura, y.shao, d.gunduz} @imperial.ac.uk.

A. Perotti and B. Popovic are with the Radio Transmission Technology Lab, Huawei Technologies Sweden AB, Kista 164-94, Sweden. Emails: {alberto.perotti, branislav.popovic}@huawei.com

was introduced and studied by Shannon [1]. In general, the formulation of communication with feedback involves a transmitter-receiver pair connected via a forward and a feedback channel, and the goal is to reliably deliver a block of bits from the transmitter to the receiver with the help of feedback. Shannon investigated the impact of feedback on the forward channel capacity by assuming perfect channel output feedback with unit delay. He proved an important result that the classical capacity of a memoryless forward channel does not increase in the presence of feedback [1].

While feedback does not increase the capacity, it is known to simplify the communication scheme and improve the reliability in the finite block length regime. Most practical communication systems employ feedback, either in the form of channel state information feedback, or automatic repeat requests (ARQs). While the former allows adaptation to channel variations, the latter increases reliability by adjusting the block length according to the noise realization. Another method to exploit feedback to increase reliability was introduced by Schalkwijk and Kailath in [2] and [3]. In the classical Schalkwijk-Kailath (SK) scheme, the transmitter encodes the message using pulse amplitude modulation (PAM) initially, and subsequently refines its estimate at the receiver in an iterative manner by sending a scaled version of the residual error at each iteration. Provided that the transmission rate is below the capacity, the SK scheme achieves a double exponential decay of the decoding error probability with the increase in code length. Designing coding and modulation schemes that can best exploit the feedback has been an ongoing challenge over decades [1]–[13], yielding a significant impact on a variety of applications that require ultra-reliable short-packet communications [14], [15], such as autonomous vehicles, industrial automation and control, tactile Internet, and augmented/virtual reality, to count a few.

Existing feedback codes can be classified as 'human-crafted' codes [2], [3], [5], [7]–[9], and deep learning (DL)-aided codes [10]–[13]. Among human-crafted codes, two notable works are the SK scheme [2], [3], [6] and its extension to the *active feedback* scenario, the modulo-SK scheme [9]. Here, active feedback refers to the scenario in which the feedback symbols can also be encoded by the receiving terminal prior to transmission to the transmitter.

A main disadvantage of the aforementioned human-crafted codes is that they are sensitive to numerical precision and quantization errors [6], [8], [10], [13]. Since the message is mapped to a 2^K-ary PAM constellation, the number of bits required to represent all the statistics in this process grows linearly with K. When K is large, these schemes suffer

from severe quantization errors caused by the finite-precision arithmetic and finite quantization levels of the electronic parts and components, e.g., power amplifier and FPGA chip. On the other hand, DL-aided feedback codes model the communication system as an autoencoder [10]–[13], in which the encoder and decoder are modeled as a pair of deep neural networks (DNNs), while the wireless channel is treated as an untrainable stochastic layer. The code is obtained by end-to-end unsupervised learning to minimize the reconstruction error of the block of bits at the receiver.

Compared with human-crafted codes, DL-based feedback codes do not suffer from the constraint of finite precision and quantization levels, as they can be trained with such constraints embedded into the training process. Moreover, they are very flexible and can be easily trained for different scenarios. Specifically, both the SK and modulo-SK schemes are designed for the setup of unit-time delayed feedback and AWGN channels with a specific pair of feedforward and feedback signal-to-noise ratios (SNRs). In contrast, DL-based codes can be easily generalized to more practical scenarios [10], [13], such as feedback with greater delays, block feedback, as well as non-Gaussian noise or fading channels. On the other hand, existing DL-aided feedback codes suffer from the following limitations that we address in this paper:

- *Communication overhead:* In practice, each round of feedback subsequent to the use of the forward channel introduces an overhead and additional delay independent of the number of transmitted bits. We quantify the corresponding communication overhead as the number of "switches" at the source node between transmitting parity symbols and receiving feedback symbols, or equivalently the number of communication rounds T. In the previous designs, T scales linearly with the number of message bits K. One of our key objectives is to reduce this communication overhead without sacrificing performance significantly.
- *Limited set of feasible rates:* Existing schemes are limited to code rates of $1/k, k \in \mathbb{Z}^+$. Hence, another important goal of this work is to present a design that can transmit at a wider range of rates. The flexibility in the communication rate is important to achieve higher spectral efficiencies, particularly in the higher SNR regimes.
- *Lack of structure:* Existing codes are defined through the employed DNN architecture. Instead, we would like to provide a holistic view of the problem and introduce a generalized modular design, where modules can be added/removed, and implemented through arbitrary architectures addressing different requirements in terms of performance and complexity.

In this paper, we introduce the generalized block attention feedback (GBAF) code, which addresses all of the afore-mentioned limitations of existing designs. In particular, in the GBAF architecture, we introduce a novel sequence-to-sequence encoding framework. We then group the message bits into blocks, and treat each block as the information unit to be communicated. We employ the popular transformer-based encoder architecture [16]–[18] as the core sequence-to-
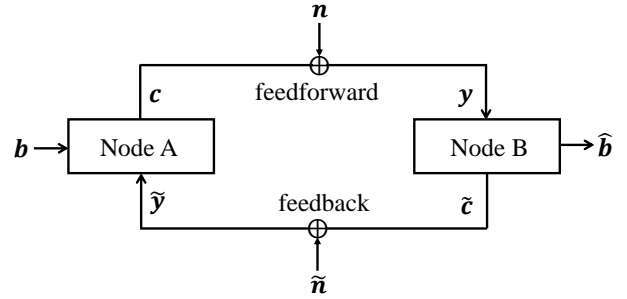

Fig. 1: Communication with block feedback: the system model.

sequence encoder module. We show that GBAF codes achieve orders of magnitude improvements in terms of the block error rate (BLER) performance over the whole range of channel SNRs compared to existing DL-based codes in the literature. Apart from [9], feedback codes in the literature are designed for a passive feedback scenario; that is, the feedback signal is simply a noisy version of the signal received at the receiver. While we also consider passive feedback in this paper, our design can be easily extended to active feedback.

The rest of the paper is organized as follows. We present the problem formulation in Section II, and provide a detailed overview of the existing feedback code structures and their limitations. The structure and modules of the GBAF code are introduced in Section III. Numerical results illustrating its superiority are presented in Section IV. We conclude the paper in Section V.

Notations – We use bold, capital bold, and capital calligraphic fonts to denote vectors, matrices, and sets, respectively, i.e., $\boldsymbol{v}$, $\boldsymbol{V}$, and $\mathcal{V}$. We use the notation $\boldsymbol{v}_{[\ ]}$, $\boldsymbol{V}_{[\ ,\ ]}$ to denote index slicing. We use the superscript for a vector/matrix/list to refer to its realization at a particular time/iteration. Finally, we use subscripts to emphasize a particular element of a sequence; for example, given a sequence of vectors $\mathcal{Q} = \{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_K\}$, $\boldsymbol{q}_i$ is used to represent the ith vector in the sequence. A list of all the notations used throughout the paper is provided in Table I.

II. PROBLEM STATEMENT

A. System model

We consider a point-to-point communication scenario with one transmitter and one receiver, as shown in Fig. 1. The objective of the transmitter is to send K bits of information, $\boldsymbol{b} = [b_1, \ldots, b_K] \in \{0,1\}^K$, to the receiver in N channel uses. We impose a rate constraint of R, that is, $K/N \geq R$. Here, we use $\boldsymbol{c} = [c_1, \ldots, c_N] \in \mathbb{R}^N$ to denote the sequence of transmitted symbols over the forward channel. We model both the forward and feedback channels as AWGN channels with independent noise terms.

We consider a feedback channel model consisting of multiple communication rounds, where in each round the transmitter transmits a vector of symbols, after which it receives a vector of feedback symbols corresponding to the transmitted symbols over the forward channel. In the literature, the channel output feedback, noiseless or noisy, is assumed to be available

TABLE I: Notations

Notation	Description
$\boldsymbol{b}$	Input bit-stream
K	Length of the bit-stream
N	Codeword length
R	Transmission rate
m	Length of message blocks
l	Number of message blocks: $l = K/m$
T	Number of interactions (communication rounds)
τ	Index of communication rounds
$\boldsymbol{c}^{(\tau)}$	Coded symbols in forward direction in communication round τ
$\boldsymbol{y}^{(\tau)}$	Received channel output at receiver in communication round τ
$\tilde{\boldsymbol{y}}^{(\tau)}$	Received channel feedback at transmitter in communication round τ
$\boldsymbol{n}^{(\tau)}$	Noise vector at forward direction in communication round τ
$\tilde{\boldsymbol{n}}^{(\tau)}$	Noise vector at feedback direction in communication round τ
$\boldsymbol{q}^{(\tau)}$	Knowledge vector at transmitter in communication round τ
$\tilde{\boldsymbol{q}}^{(\tau)}$	Knowledge vector at receiver in communication round τ
$\hat{\boldsymbol{b}}$	Predicted bit-stream at decoder
$\boldsymbol{q}_i^{(\tau)}$	Knowledge vector at the parity network for the ith message block in communication round τ
$\boldsymbol{f}_i^{(\tau)}$	Extracted feature vector at the parity network for the ith message block in communication round τ
$\boldsymbol{v}_i^{(\tau)}$	Final latent representation at the parity network for the ith message block in communication round τ
$\boldsymbol{c}_i^{(\tau)}$	Vector of coded symbols generated by the parity network for the ith message block in communication round τ
$\tilde{\boldsymbol{q}}_i^{(\tau)}$	Knowledge vector at the belief network for the ith message block in communication round τ
$\tilde{\boldsymbol{f}}_i^{(\tau)}$	Extracted feature vector at the belief network for the ith message block in communication round τ
$\tilde{\boldsymbol{v}}_i^{(\tau)}$	Final latent representation at the belief network for the ith message block in communication round τ
$\tilde{\boldsymbol{b}}_i^{(\tau)}$	Prediction of the belief network for ith message block in communication round τ
$\hat{\boldsymbol{q}}_i$	Knowledge vector at the decoder network for the ith message block.
$\hat{\boldsymbol{f}}_i$	Extracted feature vector at the decoder network for the ith message block
$\hat{\boldsymbol{v}}_i$	Final latent representation at the decoder network for the ith message block
$\boldsymbol{w}_i$	Logit vector generated at the decoder network for the ith message block

instantly at the encoder (i.e., with unit delay). However, in practice, these feedback symbols need to be encoded and/or modulated, and in general, encoding/ decoding operations, as well as the additional exchange of control information between the transmitter and receiver for every forward and feedback packet will introduce additional overheads. Hence, it is desirable to utilize feedback while introducing minimum overhead. Therefore, our goal will be to achieve the desired level of reliability with minimal number of interactions, i.e., communication rounds, between the transmitter and the receiver.

Our model would be particularly relevant in the active feedback scenario, where the feedback symbols are encoded by the receiver. In the case of passive feedback, considered in this paper, we use this model to quantify the potential overheads due to processing of the feedback symbols and generating the transmitted symbols over the forward channel based on the received feedback.

Let τ denote the index of the *communication round*. In communication round τ, the transmitter sends N_τ symbols, denoted by $\boldsymbol{c}^{(\tau)}$, in the forward direction,[1] and receives N_τ symbols[2], denoted by $\tilde{\boldsymbol{y}}^{(\tau)}$, over the feedback link, for $\tau = 1, \ldots, T-1$. We have $\sum_{\tau=1}^{T} N_\tau \leq N$. We remark that, existing schemes as well as the proposed design utilize equal-

length vectors over τ, where a slight modification appears in the systematic code design used in the previous works [10]–[12], which we will explain later.

Communication is terminated when the receiver receives $\boldsymbol{c}^{(T)}$. The received vector of symbols at the forward and feedback links, denoted by $\boldsymbol{y}^{(\tau)}$ and $\tilde{\boldsymbol{y}}^{(\tau)}$, respectively, are given by

$$\boldsymbol{y}^{(\tau)} = \boldsymbol{c}^{(\tau)} + \boldsymbol{n}^{(\tau)}, \quad \text{for } \tau = 1, \ldots, T, \tag{1}$$

and

$$\tilde{\boldsymbol{y}}^{(\tau)} = \boldsymbol{y}^{(\tau)} + \tilde{\boldsymbol{n}}^{(\tau)}, \quad \text{for } \tau = 1, \ldots, T-1, \tag{2}$$

where $\boldsymbol{n}^{(\tau)}, \tilde{\boldsymbol{n}}^{(\tau)} \in \mathbb{R}^{N_\tau}$ are the noise vectors consisting of independent and identically distributed (i.i.d.) zero-mean Gaussian random variables with variances σ_{ff}^2 and σ_{fb}^2, respectively.

If we consider T communication rounds in the forward direction, this implies that the direction of communication is switched $2T - 2$ times, which we use to quantify the overhead of the feedback mechanism. As mentioned above, larger T corresponds to more overhead.

The focus of our paper is to design a mechanism for generating symbols in forward and feedback directions for each communication round τ. Before describing the particular encoding mechanism we propose, we introduce the so-called *knowledge vectors*, $\boldsymbol{q}^{(\tau)}$ and $\tilde{\boldsymbol{q}}^{(\tau)}$, which refer to all the available information at the transmitter and the receiver, respectively, when generating the symbols transmitted in communication round τ. The knowledge vector at the transmitter,

---

[1]We consider discrete-time analog transmission [19].

[2]In general, in the active feedback scenario, we can have a different number of symbols transmitted over each communication round of the forward and feedback channels. Here, we set them to be equal as we assume that the symbols transmitted over the feedback channel are simply the symbols received by the receiver, i.e., passive feedback.

$q^{(\tau)}$, consists of the original bit stream, previously transmitted symbols, and the received feedback symbols up to round τ, i.e.,

$$q^{(\tau)} = [b, c^{(1)}, \ldots, c^{(\tau-1)}, \tilde{y}^{(1)}, \ldots, \tilde{y}^{(\tau-1)}]. \quad (3)$$

The knowledge vector at the receiver consists of the received channel outputs up to round τ

$$\tilde{q}^{(\tau)} = [y^{(1)}, \ldots, y^{(\tau)}]. \quad (4)$$

Let $M^{(\tau)}$ denote the encoding function at the transmitter, where $M^{(\tau)}(q^{(\tau)}) = c^{(\tau)} \in \mathbb{R}^{N_\tau}$. Once the transmission of all the symbols is completed, a decoding function D is employed at the receiver to recover the original bit stream, i.e., $\hat{b} \in \{0,1\}^K = D(\tilde{q}^{(T)})$.

The code must satisfy an average power constraint on the transmitted symbols:

$$\mathbb{E}\left[\frac{1}{N}\sum_{\tau=1}^{T}\langle c^{(\tau)}, c^{(\tau)}\rangle\right] \leq 1. \quad (5)$$

Hence, the SNR in the forward direction is given by $SNR_{ff} = 1/\sigma_{ff}^2$, while the SNR in the feedback channel is $SNR_{fb} = 1/\sigma_{fb}^2$. We refer to the case $\sigma_{fb}^2 = 0$ as *noiseless feedback*.

Remark (Systematic codes)**.** *We refer to a feedback code as a systematic feedback code, if there is an initial stage at $\tau = 0$, at which the encoder maps the original bit stream to its BPSK modulated version, i.e., $N_0 = K$, and $M^{(0)}(b) = c^{(0)} = \alpha(2 \cdot b - 1)$, where α is chosen to satisfy the power constraint.*

$$M^{(0)} : q^{(0)} = b \xrightarrow{BPSK} c^{(0)} = \bar{b} = \alpha(2 \cdot b - 1). \quad (6)$$

We note that, an additional iteration index $\tau = 0$ is allocated for the systematic code part to be able to align different DL-based feedback code designs. Independent from the employment of the systematic part, DL-based symbol encoding starts from $\tau \geq 1$.

Although we have restricted the above definition to BPSK modulation for the sake of simplicity, the same notion can be extended to other modulation schemes with larger constellations. In general, there is no particular reason to restrict ourselves to a systematic feedback scheme, but we defined this set of codes explicitly as the DL-based codes considered in the literature [10]–[12] are all systematic codes.

B. Existing DL-Based Feedback Codes

The ultimate challenge in feedback codes is designing an iterative encoding process for the parity symbols at the transmitter, and a decoding process for the received symbols at the receiver. DNN-based feedback codes aim to tackle this issue by considering the encoding and decoding mappings, M and D, respectively, as DNN architectures, and by training them for a sufficient number of randomly generated bit streams to achieve the final network model/weights. It has been shown that such an end-to-end training approach is highly effective for designing feedback codes [10], [12], [13]. Now we revisit some of the existing feedback code designs

in the literature and illustrate how they operate according to our generic framework.

1) General overview: Existing DL-based feedback codes in the literature, DeepCode [10], DEF [11], and DRF codes [12] consider systematic and passive feedback schemes. Communication process is divided into two phases, $\tau = 0$ and $\tau > 0$. $M^{(0)}$ corresponds to the systematic modulation scheme described in (6). In the second phase, $\tau > 0$, a DNN architecture, H_{encoder}, is used to generate the vector of parity symbols:

$$H_{\text{encoder}} : S_{\text{encoder}}(q^{(\tau)}) \xrightarrow{Neural-encoder} c^{(\tau)}, \quad (7)$$

where $S_{\text{encoder}}(\cdot)$ denotes the pre-processing function that defines how the knowledge vector $q^{(\tau)}$ is fed to the DNN architecture H_{encoder}.

2) Sequence-to-one encoding: Although the existing DL-based encoder designs employ different NN architectures, see Table II, they often follow the same structure for processing the knowledge vector $q^{(\tau)}$ in order to generate channel symbols. Function $S_{\text{encoder}}(\cdot)$ is used to transform the knowledge vector into a sequence of vectors that can be fed to the DL-based encoder. Hence, for the encoding process, $q^{(\tau)}$ is first transformed into a sequence of vectors $\left\{q_1^{(\tau)}, \ldots, q_K^{(\tau)}\right\}$, whose length is equal to the length of the original bit-stream, which is then fed to the network to generate a vector of channel symbols.

The encoding strategy, followed in the previous code designs, simply assumes that $q_i^{(\tau)}$ is the knowledge vector at round τ corresponding to the ith bit of the original bit-stream. The employed sequence-to-one encoding approach has two distinguishing features. First, at any communication round $\tau = nK + i$, during the $n + 1$th pass over the bit-stream, they generate one vector of symbols $c^{(\tau)}$ that corresponds to a particular knowledge vector $q_i^{(\tau)}$; hence, when the corresponding feedback is received at the transmitter, only $q_i^{(\tau)}$ is updated to obtain $q_i^{(\tau+1)}$ before the next vector of symbols, $c^{(\tau+1)}$, is generated. Second, the encoding process is causal; that is, for generating $c^{(\tau)}$, $\tau = nK + i$, only the knowledge vectors $\left\{q_1^{(\tau)}, \ldots, q_i^{(\tau)}\right\}$ are utilized, simply the ones whose index is larger then i are ignored. We illustrate the overall sequence-to-one encoding process for a particular $\tau \geq 1$ in Fig. 2.

While we provided above the general structure of the existing DL-based feedback code designs, they all use a special case of this general form with a single pass over the bit-stream, i.e., $n = 1$, and exactly two symbols are generated at each iteration, i.e., $N_\tau = 2$ for all $\tau \geq 1$ while systematic encoding with BPSK modulation is used for $\tau = 0$. For this particular setup, one achieves the rate $R = 1/3$ by using $K+1$ communication rounds in the forward direction. In general, for given K, R, and N_τ, $T = \frac{K}{N_\tau} * \left(\frac{1}{R} - 1\right) + 1$ communication rounds are required in the forward direction[3].

---

[3]To prevent any confusion, we ignore the extra zero padding strategy introduced in [10].
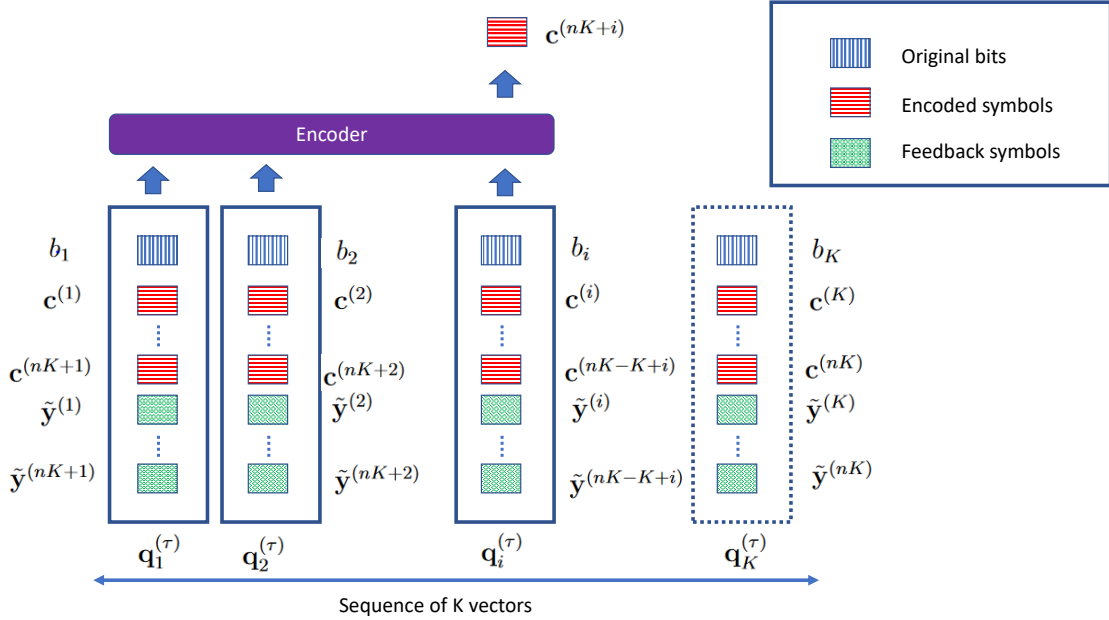
Fig. 2: Visualisation of the sequence-to-one encoding approach at iteration $\tau = nK + i$. The knowledge vector $\boldsymbol{q}^{(\tau)}$ is divided into K, where $\boldsymbol{q}_i^{(\tau)}$ corresponds to the knowledge vector for the i-th message bit. Each channel input vector $\boldsymbol{c}^{(\tau)}$ corresponds to a particular message bit i, and is generated using only the knowledge vectors corresponding to message bits $1, \ldots, i$. The transmitted symbol and the corresponding channel output feedback are then added to the knowledge vector of bit i, $\boldsymbol{q}_i^{(\tau)}$, to be used in the generation of future channel symbols.

Design	H_{decoder}	H_{encoder}
DeepCode [10]	Bi-GRU	GRU [20]
DRF Code [12]	Bi-LSTM	LSTM [21]
AttentionCode [13]	Transformer Encoder	Transformer Encoder [16]

TABLE II: DNN-based designs for feedback codes.

Similarly, at the receiver, a combination of DNN architecture H_{decoder} and pre-processing function $S_{\text{decoder}}(\cdot)$ is used as the decoding function D, i.e.,

$$H_{\text{decoder}}\left(S_{\text{decoder}}\left(\tilde{\boldsymbol{q}}^{(T)}\right)\right) \tag{8}$$

$$= H_{\text{decoder}}\left(S_{\text{decoder}}\left([\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(T)}]\right)\right) \tag{9}$$

$$\xrightarrow{Neural-decoder} \hat{\boldsymbol{b}} \in \{0,1\}^K. \tag{10}$$

The particular DNN architectures employed, both at the encoder and the decoder, in the existing feedback codes proposed in the literature are listed in Table II.

III. GENERALIZED BLOCK ATTENTION FEEDBACK (GBAF) CODES

Following the general design principles summarized above, the common aspect of the existing DNN-based feedback codes is to consider the given bit-stream as a sequence and utilize DNN architectures that are particularly designed for processing sequences, such as the long short-term memory (LSTM) and gate recurrent unit (GRU) architectures, to generate parity symbols as well as to decode them to recover the original bit stream. The proposed GBAF code design differentiates itself

from the existing codes in several aspects. Below we present the architecture of the GBAF codes, and emphasize its main novelties with respect to the state-of-the art.

A. Overview of Innovations

1) Sequence-to-sequence encoding: The key novelty of the GBAF code design, different from the existing strategies, is the way the sequence is processed at the encoder to generate channel symbols. While the existing strategies follow the *sequence-to-one* coding principle mentioned above, the GBAF code uses *sequence-to-sequence* encoding to generate a vector of symbols corresponding to the whole message bits. The knowledge vector at the encoder, $\boldsymbol{q}^{(\tau)}$, is first transformed into a sequence of K vectors $\left\{\boldsymbol{q}_1^{(\tau)}, \ldots, \boldsymbol{q}_K^{(\tau)}\right\}$. However, unlike sequence-to-one coding, at each communication round τ, parallel processing of these knowledge vectors is used instead of causal processing; that is, for each knowledge vector $\boldsymbol{q}_i^{(\tau)}$, $i = 1, \ldots, K$, a vector of coded symbols $\boldsymbol{c}_i^{(\tau)}$ is generated simultaneously and the transmitted codeword $\boldsymbol{c}^{(\tau)}$ is obtained by concatenating these vectors, i.e., $\boldsymbol{c}^{(\tau)} = \left[\boldsymbol{c}_1^{(\tau)}, \ldots, \boldsymbol{c}_K^{(\tau)}\right]$. Accordingly, when the corresponding feedback becomes available at the transmitter, all the elements of the sequence of knowledge vectors are updated simultaneously. Hence, the number of interactions between the receiver and the transmitter does not scale with the length of the sequence, but with the coding overhead, which is the inverse of the rate.

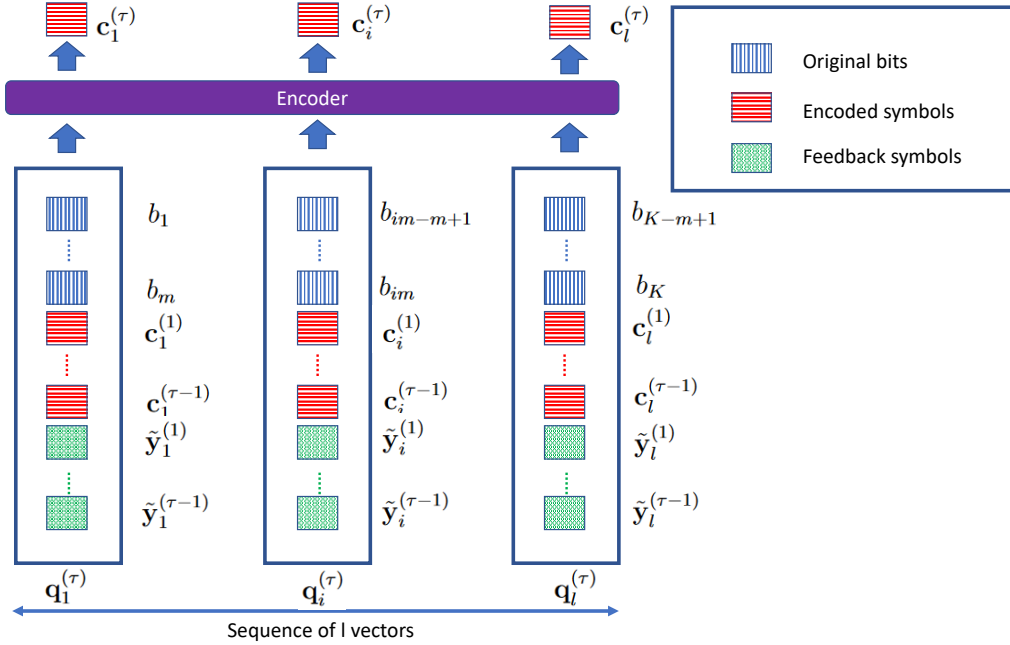*2) Sequence of bits to sequence of blocks:* Although the parallel execution with sequence-to-sequence encoding

Fig. 3: Visualisation of the sequence-to-sequence encoding approach with block formation for a block size of m, where $l = K/m$, at iteration τ. Bits are grouped into l blocks, each consisting of m bits. The knowledge vector $\boldsymbol{q}^{(\tau)}$ is also divided into l, where $\boldsymbol{q}_i^{(\tau)}$ corresponds to the knowledge vector about the l-th block of message bits. Each generated channel symbol vector $\boldsymbol{c}_i^{(\tau)}$ corresponds to a particular block of message bits $b_{im-(m-1)}, \ldots, b_{im}$, but all the available knowledge vectors are used simultaneously to generate all the channel input vectors in iteration τ. The transmitted symbol and the corresponding channel output feedback are then added to the knowledge vectors of all the message blocks.

reduces the communication overhead compared to existing frameworks, the limitation of the number of feedback iterations by the rate leads to under-utilization of the feedback mechanism. Apart from that, we identify three other limitations:

1) performing sequence-to-sequence encoding for large sequences is computationally expensive and has large memory requirements;
2) when the number of feedback iterations is limited, information gathered for each element of the sequence is also limited;
3) sequence-to-sequence encoding alone does not offer a wider range of rate options compared to former designs.

We highlight that, in the existing DL-based feedback codes, the length of the processed sequence is equal to the length of the bit-stream; that is, each element of the sequence corresponds to a single message bit and the corresponding channel input and feedback symbols. Hence, to address all the aforementioned limitations, we divide the bit-stream into group of bits, which we refer to as a *message block*. As a result, each element of the sequence corresponds to a block of bits and the corresponding transmitted channel input and the feedback symbols, which leads to a reduction in the sequence length by the message block size. Then, sequence-to-sequence encoding is performed on the sequence of message blocks and the corresponding knowledge vectors. Fig. 3 illustrates the sequence-to-sequence encoding scheme with message blocks. For sequence-to-sequence encoding, use of blocks of bits

instead of single bits reduces the number of feedback iterations for a fixed code rate by the block size, which reduces the feedback overhead. Additionally, the achieved reduction on the length of the sequence also reduces the computational complexity and the memory requirements.

Moreover, for the specific transformer-based DNN architecture we employ for our code design, when the elements of each sequence corresponds to a block of bits rather than a single bit, one can obtain a more informative embedding for each element of the sequence, which improves the information processing capability of the transformers. This will become more clear when we introduce the details of the architecture below.

More formally, GBAF code design divides the K original information bits into l blocks of m bits each. Here, we assume m divides K, such that $K = l \cdot m$. These form our initial l knowledge vectors (see Fig. 3). We utilize sequence-to-sequence encoding at each round of communication treating the l knowledge vectors as the input sequence, and generate the channel symbols corresponding to each message block, equivalently, to each knowledge vector. Then, we update the sequence of knowledge vectors with the transmitted symbols and the received feedback symbols, by appending them to the corresponding knowledge vectors. A total of $N_\tau = l = K/m$ symbols, one parity symbol for each knowledge vector, are transmitted at each round τ. One can observe that, given rate R and block size m, the number of required communication rounds is $T = m/R$, which does not scale with K. Furthermore, by choosing different $T \in \mathbb{Z}^+$ and $m \in \mathbb{Z}^+$ values it
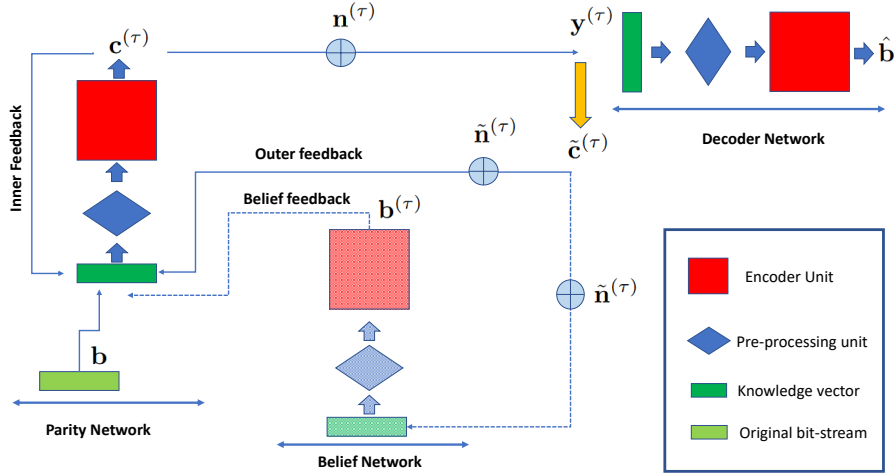
Fig. 4: Illustration of the overall GBAF code architecture. The green, blue and red blocks denote the knowledge vector, pre-processing unit, and encoder unit, respectively. The dashed lines and shapes indicate the units and connections that are optional.

is possible to obtain a wide range of code rates, $R = m/T$. Hence, the rate of the code can be adjusted by changing the block size m and the number of communication rounds T, which is also equivalent to the total number of parity symbols transmitted per block. From the encoding processes illustrated in Fig. 2 and Fig. 3, one can also observe that, under the same rate constraint, the sequence-to-sequence encoding approach requires l times fewer interactions between the receiver and the transmitter compared to the sequence-to-one approach, which results in a reduced feedback overhead in practice, as argued above.

So far, we have identified two novel aspects of the GBAF code design; namely, i) utilizing sequence-to-sequence encoding instead of sequence-to-one encoding, and ii) reorganizing the sequence into blocks of bits before encoding. These general design principles are independent from the particular DNN architecture, and can be combined with any architecture that can be adapted for sequence-to-sequence encoding. The third novel aspect of the GBAF code is its architecture and the newly introduced modules. Different from the existing designs, we employ a novel *transformer* encoder architecture for encoding and decoding. Furthermore, we introduce custom modules, such as a feature extractor to deal with large noise realizations. In order to provide a more holistic view from the design perspective, below we present the GBAF code design in three parts: general architecture, specific modules, and implementation.

B. GBAF Architecture

From an operational point of view, we employ two types of components in the overall design, namely an encoder unit and a pre-processing unit. Motivated by the SK scheme [2], the transmitter consists of two cascaded networks, each of which consists of a pre-processing unit followed by an encoder unit. We refer to the initial network as the *belief network* and the latter as the *parity network*. The objective of the belief network is to generate a belief on the predicted bits at the receiver,

while the objective of the *parity network* is to generate parity symbols to improve the prediction accuracy at the receiver. The receiver also employs a network with the same structure to predict the original bit stream, which we refer to as the decoder network. In the overall architecture, we identify three types of information flows, which we call as different feedback mechanisms:

- **Inner Feedback:** We use the term inner feedback to refer to a feedback mechanism within each network. It is used for the encoder network to recall the previously generated parity symbols.
- **Belief Feedback:** The belief feedback refers to the information flow from the belief network to the encoder network.
- **Outer Feedback:** The outer feedback is the physical feedback signals from the receiver to the transmitter.

The overall architecture is illustrated in Fig. 4. In the introduced architecture, the belief network and the belief feedback are optional, and they can be added or removed as desired, presenting a trade-off between complexity and performance, and the objective of using two networks is to disentangle the task of generating parity bits and predicting the belief at the receiver. However, by bypassing the belief network and disabling the belief feedback, both tasks can be fulfilled by the parity network.

C. Modules

In the GBAF code, for all the encoder units we utilize the same DNN architecture denoted by H_{encoder}, which simply maps sequences of l vectors of size d_{in} to sequences of vectors of size d_{out} with the same length, i.e., $H_{\text{encoder}}(\boldsymbol{q}_1, \ldots, \boldsymbol{q}_l) = \mathcal{U} = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_l\}$, such that $\boldsymbol{q}_i \in \mathbb{R}^{d_{in}}$, $\boldsymbol{u}_i \in \mathbb{R}^{d_{out}}$. H_{encoder} unit consists of three modules: feature extractor H_{extract}, sequence-to-sequence encoder H_{s2s}, and output mapping H_{map}. Accordingly, $H_{\text{encoder}} = H_{map} \circ H_{s2s} \circ H_{\text{extract}}$, where $\circ$ denotes composition. The end-to-end architecture of

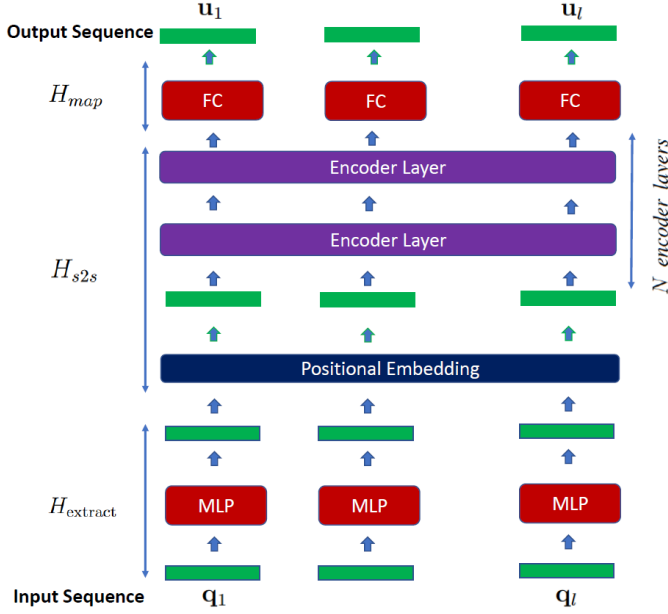Fig. 5: Illustration of an encoder unit, H_{encoder}.

the encoder unit is illustrated in Fig. 5. Below we explain each of these components in detail.

1) Feature extractor: The role of the feature extractor is to map the collected raw data for each message block to a certain vector representation, similarly to the vector embedding approach in NLP tasks [22]–[24], where the objective is to represent the words in the form of a vector and the corresponding representation inherits certain contextual information regarding the word. However, our problem has two unique challenges: i) time-evolving nature of data; and ii) the randomness in the input. By the randomness, we refer to the random noise realization at each communication round. In principle, encoder module utilizes the noise realizations in the past to generate the parity symbols; nevertheless, the outlier noise realizations, particularly in the low SNR regime, might be overemphasized when a simple linear mapping is used for feature extraction. Hence, our ultimate aim is to design a feature extractor module in a way that the impact of each raw data on the corresponding representation is limited. To this end, we utilize a multi-layer perceptron (MLP) architecture. As detailed in Appendix A, the feature extractor consists of three linear layers with two activation functions in between. The activation function can be Gaussian error linear unit (GeLU) [25] or rectified linear unit (ReLU). We use $H_{extract}^{parity}$, $H_{extract}^{belief}$, and $H_{extract}^{decoder}$ to denote the feature extractors for parity network (line 14 of Algorithm 1), belief network (line 7 of Algorithm 1), and decoder network (line 6 of Algorithm 2), respectively.

2) Sequence-to-sequence encoder: Sequence-to-sequence encoder H_{s2s} is a DNN architecture, where the sequence of early feature representations are mapped to a sequence of final latent representations by seeking certain correlations among the elements of the input sequence. The input to the H_{s2s} is a sequence of d_{model} dimensional vectors of length l and the output is again a sequence of d_{model} dimensional vectors of

length l. Hence, a wide range of existing DNN architectures, particularly those employed for natural language processing (NLP) tasks, such as LSTM, GRU, transformer, can be utilized as H_{s2s}. We use H_{s2s}^{parity}, H_{s2s}^{belief}, and $H_{s2s}^{decoder}$ to denote the sequence-to-sequence encoder for parity network (line 15 of Algorithm 1), belief network (line 8 of Algorithm 1), and decoder network (line 7 of Algorithm 2), respectively.

We have observed that the transformer architecture performs particularly well for sequence-to-sequence encoding. Hence, for H_{s2s}, we consider sequence of N *encoder layers* of the transformer architecture[4], which consists of three main components: the feed-forward module, the multi-head self-attention module, and the layer normalization module as illustrated in Fig. 6. Next, we briefly explain the structure of the attention and feed-forward modules.

Self-Attention Module: Self-attention mechanism is the key enabler for extracting relative information from a sequence. The core idea of the attention mechanism is to utilize a set of *key-value* pairs, for a given query, to generate an output. Consider d_k, d_q, d_v dimensional vectors of key, query, and value, respectively, and N queries, K keys and K values, represented in a stacked form as $\boldsymbol{Q} \in \mathbb{R}^{N \times d_k}$, $\boldsymbol{K} \in \mathbb{R}^{K \times d_k}$, $\boldsymbol{V} \in \mathbb{R}^{K \times d_u}$, respectively. The objective of the attention mechanism is to obtain the weights required for combining the set of values to provide an output. The underlying mechanism used is scaled dot-product attention, i.e.,

$$Attn(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \underbrace{Softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)}_{\boldsymbol{W} \in \mathbb{R}^{N \times K}}\boldsymbol{V}, \qquad (11)$$

where $\sqrt{d_k}$ is used to normalize the output of the dot-product before the softmax layer. The multi-head attention follows the same principle, but query, key and value vectors are first processed through a linear layer and fed to multiple attention mechanisms/heads, which are executed simultaneously. Then the final output is obtained by concatenating the outputs of all the attention mechanisms/heads.

Feed-Forward Module: The feed-forward module consists of two fully connected layers with a non-linearity (activation) between them, which can be formally described as

$$FFN(\boldsymbol{x}) = \phi(x\boldsymbol{W} + \boldsymbol{d})\tilde{\boldsymbol{W}} + \tilde{\boldsymbol{d}}, \qquad (12)$$

where $\phi(\cdot)$ denotes a non-linear activation function, such as ReLU or GELU, $\boldsymbol{W}$ and $\tilde{\boldsymbol{W}}$ denotes matricies for linear projection, and, $\boldsymbol{d}$ and $\tilde{\boldsymbol{d}}$ denotes the vector for bias terms. Given an input vector of size d_{model}, first linear layer increases the model size to $\delta \times d_{model}$, which is later reduced to d_{model} again by the second linear layer. Here, δ is often called the *scaling factor*. It has been argued that the feed-forward module functions as a memory [26]. In our implementation, we set $d_{model} = 32$, consider a single attention head, and set $\delta = 4$ at the feed-forward module following the common implementation [17], and finally for the layer normalization we follow the pre-layer normalization option [27]. For the

---

[4]We follow the standard implementation used in the Pytorch library: https://pytorch.org/docs/stable/_modules/torch/nn/modules/transformer.html#TransformerEncoderLayer
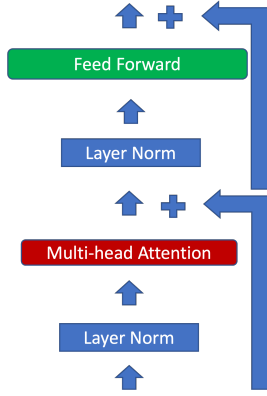
Fig. 6: Visualization of the encoder layer.

number of encoder layers N, we consider $N_{parity} = 2$, $N_{\text{belief}} = 2$ and $N_{\text{decoder}} = 3$. For further details about the transformer architecture we refer the reader to [16], [17], [28] and references therein.

3) Output mapping: The output mapping H_{map} is used to map the final latent representation, obtained by the sequence-to-sequence encoder H_{s2s}, to a particular form depending on the purpose. For example, in the parity network, H_{map} maps the final representation to a parity symbol, whereas in the belief network and decoder, it is used for classification purposes. Common to all three networks, H_{map} consists of a single fully-connected layer with an input size d_{model} and output size d_{out}; however, when it is used for classification, as in the belief and decoder networks, the fully-connected layer is followed by an additional softmax layer.

Since only one parity symbol is generated per block, we consider $d_{out} = 1$ for H_{map}^{parity}. On the other hand, decoder network aims to map each block to one of the 2^m possible m-length bit streams. Hence, for H_{map}^{parity}, we have $d_{out} = 2^m$. On the other hand, for the belief network H_{map}^{belief}, we set $d_{out} = 2m$, that is for each original bit in the block we generate two values in order to represent the likelihood values $P(b_i = 0)$ and $P(b_i = 1)$ as a belief, with the help of a softmax layer[5]. Finally, we note here that due to the average power constraint, an extra layer for power normalization is required following the H_{map}^{parity}, which follows the same procedure in [10], [13].

D. Implementation and Training Procedure

Here, we illustrate how the proposed GBAF code architecture is executed from an algorithmic perspective in order to highlight its iterative structure. To describe the overall encoding procedure at the transmitter, we introduce an iterative algorithm, called unified iterative parity symbol encoding (UIPSE), that generates l symbols after each communication round, which is detailed in Algorithm 1.

To describe the final decoding mechanism at the receiver, we introduce the joint parity symbol decoding (JPSD) algorithm, where the parity symbols belonging to each block are decoded jointly, as illustrated in Algorithm 2. Different from the

[5]Here, we remark that before the softmax operation we reshape the input, i.e., $1 \times 2m \to m \times 2$.

existing feedback code designs, due to the use of the block structure, the decoder performs classification over all possible bit blocks, 2^m in total, rather than binary classification. Hence, to recover the original bit-stream we further employ a lookup-table A (line 16-18 in Algorithm 2), such that the ith row of A, $A_{[i,:]}$, corresponds to the bit-wise representation of the ith possible block.

From the training point of view, GBAF code performs a multi-class classification task. Let $x \in \{0,1\}^m$ be the m-bit block to be transmitted. Then the relation between the data x and its label $y \in [0, \ldots, 2^m - 1]$ can be formulated as

$$y = x^T z \tag{13}$$

where $z = [2^{m-1}, 2^{m-2}, \ldots, 1]^T$. The data-label pairs (x, y) are known at the transmitter. At the end of T iterations, the receiver observes $\tilde{m}$-dimensional representation of x, denoted by $\tilde{x}$, and its task is to predict y from observation $\tilde{x}$.

To generate the training data, we first generate a random sequence of bits $b \in \{0,1\}^K$, which is then divided into l blocks, each of size m bits, and assign the corresponding label for each block as described in (13). Consequently, we have $\mathcal{B} = \{(b_1, y_1), \ldots, (b_l, y_l)\}$ as the training data with corresponding labels. The generated block of bits, $\{b_1, \ldots, b_l\}$ is then fed into the encoder, and at the end of T communication iterations the decoder outputs a sequence of l 2^m-dimensional vectors, $\{w_1, \ldots, w_l\}$, as described in Algorithm 2, which are then used to predict the class of each original block of bits. We use the cross-entropy loss function defined as

$$L(W, Y) = \sum_{i=1}^{l} \sum_{c=0}^{2^m-1} -\log \frac{\exp\left(W_{[i,c]}\right)}{\sum_{c=0}^{2^m-1} \exp\left(W_{[i,c]}\right)} \cdot \mathbb{1}_{y_i \neq c} \tag{14}$$

where $Y = \{y_1, \ldots, y_l\}$ denotes the labels of the blocks in the generated sequence, and W is the sequence $\{w_1, \ldots, w_l\}$ in matrix form. When a batch of sequences are generated for training, the loss function in (14) is evaluated by taking the average loss over the batch.

IV. NUMERICAL RESULTS

In this section, we present the results of numerical experiments using the GBAF architecture and coding principles explained above[6].

A. Experiment Setup

In all the experiments, we consider a bit stream of length $K = 51$ and a block size of $m = 3$, which corresponds to $l = 17$ blocks. We consider communication in the low forward SNR regime, where the availability of feedback can be particularly effective. Specifically, we consider $SNR_{ff} \in [-1, 2]$ dB, and allow the transmission of $T = 9$ parity bits for each block in total, which corresponds to a transmission rate of $R = 3/9 = 1/3$.

For training, we use the AdamW optimizer, which is a variation of the Adam optimizer with decoupled weight decay regularization [29]. It was observed in [12] that for DNN-aided

[6]Codes are available on https://github.com/emre1925/GBAF

Algorithm 1 Unified iterative parity symbol encoding (UIPSE)

1: **for** $\tau = 1, \ldots, T$ **do** # Generate 1 parity symbol per block at each pass
2:     **Update knowledge vector**:
3:     $\boldsymbol{q}^{(\tau)} = [\boldsymbol{b}, \boldsymbol{c}^{(1)}, \ldots, \boldsymbol{c}^{(\tau)}, \tilde{\boldsymbol{y}}^{(1)}, \ldots, \tilde{\boldsymbol{y}}^{(\tau-1)}]$
4:     **if** belief feedback is enabled **then**
5:         **Pre-process knowledge vector for belief network**:
6:         $\left\{ \tilde{\boldsymbol{q}}_i^{(\tau)}, \ldots, \tilde{\boldsymbol{q}}_l^{(\tau)} \right\} = S_{\text{belief}}(\boldsymbol{q}^{(\tau)}), \;\; \tilde{\boldsymbol{q}}_i^{(\tau)} = [\tilde{\boldsymbol{y}}_i^{(1)}, \ldots, \tilde{\boldsymbol{y}}_i^{(\tau-1)}]$
7:         **Extract features**:$\tilde{\boldsymbol{f}}_i^{(\tau)} = H_{extract}^{\text{belief}}(\tilde{\boldsymbol{q}}_j^{(\tau)})$
8:         **Attention-based neural-encoding**:$\mathcal{V}^{(\tau)} = H_{\text{s2s}}^{\text{belief}}(\tilde{\mathcal{F}}^{(\tau)})$
9:         **Generate belief feedback:** $\;\; \tilde{\boldsymbol{b}}_i^{(\tau)} = H_{map}^{\text{belief}}(\tilde{\boldsymbol{v}}_i^{(\tau)})$
10:         **Pre-process knowledge vector**:$\left\{ \boldsymbol{q}_i^{(\tau)}, \ldots, \boldsymbol{q}_l^{(\tau)} \right\} = S_{parity}(\boldsymbol{q}^{(\tau)}, \tilde{\boldsymbol{b}}^{(\tau)})$
11:     **else**
12:         **Pre-process knowledge vector**:$\left\{ \boldsymbol{q}_i^{(\tau)}, \ldots, \boldsymbol{q}_l^{(\tau)} \right\} = S_{parity}(\boldsymbol{q}^{(\tau)})$
13:     **Feature extraction:**
14:     **for** $i \in [l]$ **do** $\;\; \boldsymbol{f}_i^{(\tau)} = H_{extract}^{parity}(\boldsymbol{q}_i^{(\tau)})$
15:     **Attention-based neural-encoding**:$\mathcal{V}^{(\tau)} = H_{s2s}^{parity}(\mathcal{F}^{(\tau)})$
16:     **Symbol mapping**:
17:     **for** $i \in [l]$ **do**
18:         $c_i^{(\tau)} = H_{map}^{parity}(\boldsymbol{v}_i^{(\tau)})$ # Generate 1 parity symbol for ith block
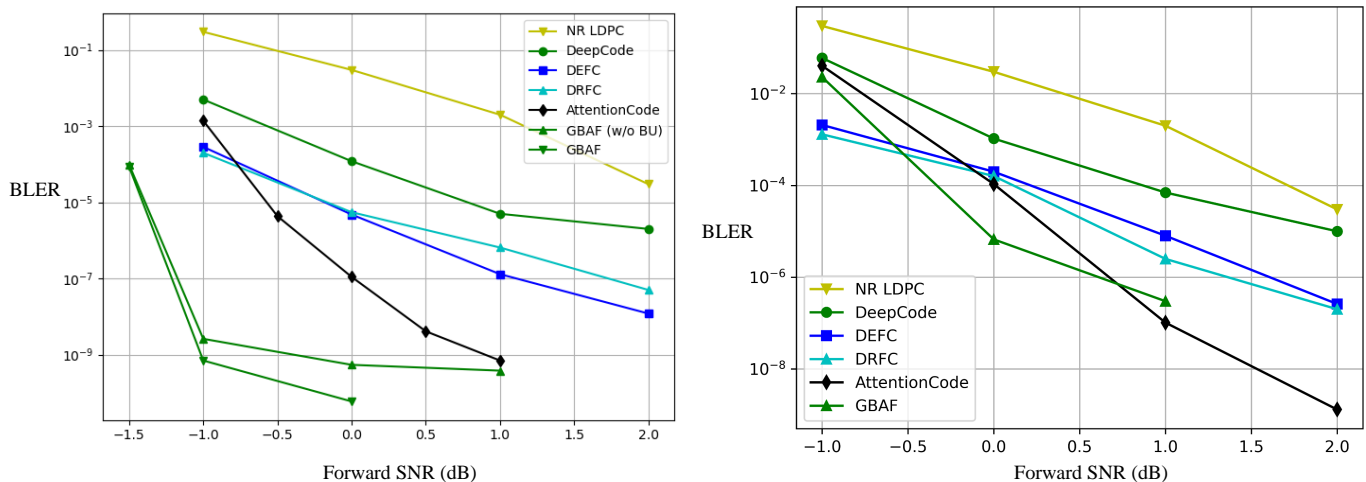

Fig. 7: Performance comparison of GBAF with AttentionCode, DEFC, DRFC, DeepCode and NR-LDPC. GBAF (w/o BU) corresponds to the GBAF architecture without the belief network. (a) Noiseless feedback; (b) Noisy feedback.

code design, the training accuracy improves with the batch size. Accordingly, we consider a batch size of $B = 8192$, the initial learning rate of 0.001, and a weight decay parameter 0.01. In addition, we apply gradient clipping with threshold 0.5. We train the network for 10^5 batches using cross-entropy loss and apply polynomial decay to the learning rate. Following the previous works, we consider the BLER as the performance measure for our analysis. To ensure the reliability of the results, during test time we wait for at least 100 block errors in each setting. [7]

[7]We note that in the regime of 10^{-9} BLER, since observing 100 block errors requires generating excessive number of samples and computational capacity, we stop the test after observing 20 errors. At the observed BLER levels of $10^{-8} - 10^{-9}$ even observing 20 errors can take several days of simulations.

We conduct our experiments under two different scenarios, noisy and noiseless feedback. In the noiseless feedback case, we use the GeLU activation function in the feature extractor, while in the noisy feedback case, we use ReLU. Further discussion on the impact of the activation function can be found in Appendix B.

B. Experimental Results

We start our analysis with the noiseless feedback scenario, i.e., $\sigma_{fb}^2 = 0$. In the first part of the simulations, we focus on a fixed transmission rate of $R = 1/3$, and compare the proposed design with the existing DNN-based feedback designs DeepCode [10], DEFC [11], DRFC [12], AttentionCode [13] as well as NR-LDPC code [30]. In the first part of the simulations, we examine two variations of the GBAF code

---

Algorithm 2 Joint parity symbol Decoding (JPSD)

1: **Update Knowledge vector**:
2: $\hat{\boldsymbol{q}} = [\tilde{\boldsymbol{c}}^{(1)}, \ldots, \tilde{\boldsymbol{c}}^{(T-1)}, \boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(T)}]$
3: **Pre-process knowledge vector for decoder network**:
4: $S_{\text{decoder}}(\hat{\boldsymbol{q}}) = \{\hat{\boldsymbol{q}}_1, \ldots, \hat{\boldsymbol{q}}_l\}, \quad \hat{\boldsymbol{q}}_i = [\tilde{\boldsymbol{y}}_i^{(1)}, \ldots, \tilde{\boldsymbol{y}}_i^{(T)}])$
5: **Feature extraction**:
6: **for** $i \in [l]$ **do** $\boldsymbol{f}_i = H_{extract}^{decoder}(\hat{\boldsymbol{q}}_i)$
7: **Attention-based neural-encoding**: $\hat{\mathcal{V}} = H_{s2s}^{decoder}(\hat{\mathcal{F}})$
8: **Mapping**:
9: **for** $i \in [l]$ **do** $\boldsymbol{w}_i = H_{map}^{decoder}(\hat{\boldsymbol{v}}_i)$
10: **Block-wise classification**: # predict the block index
11: **for** $i \in [l]$ **do** $p_i = \max_j (\boldsymbol{w}_i)_{[j]}$
12: **Block index to bit stream conversion**: # map block indices to original bits
13: **for** $i \in [l]$ **do** $\hat{\boldsymbol{b}} = [\hat{\boldsymbol{b}}, \boldsymbol{A}_{[p_i, :]}]$

---

depending on the adoption of belief network in order to highlight its impact on the performance. The BLER performance results are illustrated in Fig. 7(a) for the forward SNR values within the range of $[-1, 2]$ dB. The results clearly highlight that GBAF code provides an order of magnitude improvement compared to the best performing alternative in the literature. We also observe that the adoption of the belief network further improves the performance. Nevertheless, design of the feature extraction module becomes more critical when the belief network is employed, and we observe that the belief network with the introduced feature extraction module may not be effective for the noisy feedback scenario. For now, we consider their joint design as an open research challenge, and in the remaining simulations we disable the belief network for the GBAF code.

Next, we consider the scenario in which the feedback channel is also exposed to additive Gaussian noise with $1/\sigma_{fb}^2 = 20$dB. The illustrated results in Fig 7(b) indicate that except the lowest SNR value of -1dB, GBAF code outperforms DEFC, DRFC, DeepCode and NR-LDPC. We also observe that at higher SNR values AttentionCode may outperform the GBAF code. However, we highlight the fact that GBAF code utilizes the feedback less frequently, approximately $6\times$ less, compared to the other codes considered in the figure. Hence, we can conclude that better or similar performance can be achieved with much less overhead.

In the second set of simulations, we will highlight the flexibility of the proposed GBAF code design in terms of channel code rates it can achieve. Unlike the existing designs, the proposed framework can be easily adjusted to obtain codes at different rates by changing the number of parity symbols transmitted. This requires no variations in the architecture itself. To this end, we consider $T = 8, 7, 6, 5$ to achieve code rates $R = 3/8, 3/7, 3/6, 3/5$, respectively, and measure the BLER performance for forward SNR values in the range of $[-1, 3]$ with noiseless feedback. The BLER performance achieved with these codes is presented in Table III.

The results demonstrate that, in the higher SNR regimes, it is possible to achieve acceptable BLER values with even

higher code rates. For example, a BLER target of 10^{-5}, which is sufficient for many tasks [14], can be achieved at rates $R = 3/8, 3/7, 3/6, 3/5$ for $SNR_{ff} = 0, 1, 2, 3$ dB, respectively. Hence, unlike the existing designs, GBAF design exhibits certain flexibility for rate adaptation based on the SNR. We also notice from the table that the BLER performance degrades quickly as the code rate approaches the channel capacity at that SNR value. On the other hand, GBAF code manages to drastically lower the error rate when the code rate falls slightly below the capacity. We also note that the GBAF code performance in Fig. 7(a) saturates to a BLER of 10^{-9} above 0 dB; however, simulations at this BLER levels are less reliable as the code rarely observes any errors. Therefore, it is very unlikely to achieve BLER values lower than 10^{-9} even at higher SNRs. On the other hand, we can see in Table III that, when the code rate is increased to $R = 1/2$, the GBAF code performance does not saturate up until 3 dB.

C. Further Discussions

The performance comparison between AttentionCode and GBAF code illustrates that the performance gain achieved by the GBAF code is not only related to the chosen sequence-to-sequence encoder architecture but also to the way it is implemented. Besides, compared to the AttentionCode implementation, GBAF code reduces the computational complexity and the memory requirement since the block encoding approach induces a reduction, linearly proportional to the block-size, in the sequence length. The computational complexity of the transformer architecture is $\mathcal{O}(l^2)$, although there are recent works targeting linear complexity [31]–[33]. This implies m^2 times reduction in complexity, which makes GBAF codes more practical compared to the AttentionCode for longer block lengths. On the other hand, the computational complexity still depends quadratically on the sequence length, l. Hence, when the message length K increases, limiting the complexity of the transformer architecture may require increasing the block size m. However, this would then significantly increase the complexity of the output mapping module H_{map}^{parity}, which grows exponentially as a function of m. Therefore, adoption of a light/sparse attention mechanism to the GBAF architecture for large block length code design, which would allow processing long sequences with limited computational complexity without sacrificing the performance is an important future research direction.

We also remark that existing solutions, e.g., DeepCode or DRF code, transmit exactly two parity symbols at each communication round, thus for rate $R = 3/9$, we need $T = 52$ interactions between the forward and feedback channels, whereas GBAF requires only $T = 9$, which implies a significant reduction in the overhead. Finally, we expect that by utilizing the curriculum learning scheme used in [13], the BLER performance can be improved further, especially for higher SNR values, where we observe certain saturation in the BLER performance.

D. Fading channels

In this section, we evaluate the performance of GBAF codes over fading channels. In particular, we consider the fading

TABLE III: BLER of GBAF codes at different code rates R.

SNR/Rate	3/8	3/7	3/6	3/5
-1 dB	1.8×10^{-2}	-	-	-
0 dB	6.15×10^{-8}	2.8×10^{-3}	-	-
1 dB	2.7×10^{-8}	7.5×10^{-8}	1×10^{-2}	-
2 dB	-	1×10^{-9}	1.5×10^{-6}	6.5×10^{-2}
3 dB	-	-	2.7×10^{-8}	8.7×10^{-7}

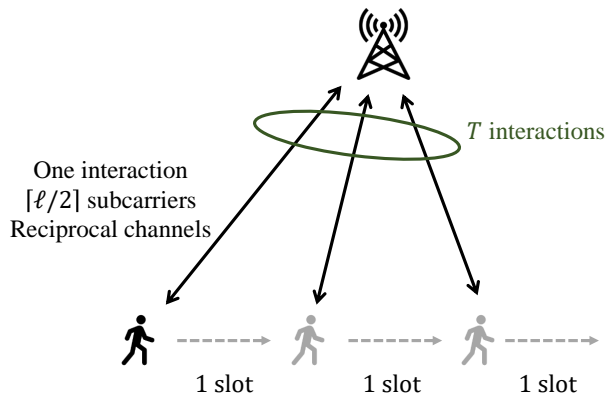Fig. 9: The PDF of the amplitude and phase obtained from the sampled channel gains of a subcarrier.

Fig. 8: The communication between a mobile user and a GNB with GBAF code. The communication lasts for T interactions, corresponding to T slots. In each interaction, the ℓ codes symbols are transmitted from the mobile user to the gNB (feedforward link) in $\lceil \ell/2 \rceil$ subcarriers; the feedback is transmitted from the gNB to the mobile user (feedback link) via the same $\lceil \ell/2 \rceil$ subcarriers, where we assume reciprocal channels in one interaction.

channel defined in new radio (NR) clustered delay line (CDL). A CDL is used to model the channel when the received signal consists of multiple delayed clusters, where each cluster contains multi-path components with the same delay but slight variations in the angles of departure and arrival.

We consider communication between a mobile user and gNodeB (gNB), denoted by node A and node B, respectively, with GBAF code. The speed of the mobile user is v_u m/s and the root-mean-square (RMS) delay spread is 100 ns. The 5G system is configured with a carrier frequency of 3.5 GHz, a subcarrier spacing of 30 KHz, and a slot duration of 0.5 ms. As shown in Fig. 8, the user communicates with the gNB in T interactions. One interaction corresponds to one slot and the ℓ real coded symbols are modulated onto $\lceil \ell/2 \rceil$ subcarriers. We assume reciprocal channels, meaning that the channel gains of the $\lceil \ell/2 \rceil$ subcarriers are the same for the uplink (from the user to the gNB) and downlink (from the gNB to the user) transmissions in one interaction.

1) Channel-gain generation: Channel gains are generated by QUAsi Deterministic RadIo channel GenerAtor (QuaDRiGa) [34] using the CDL-Model for NLOS (3GPP TR38.901 NR-CDL-C). Specifically, for a given mobile speed v_u, we generate two long move paths of the mobile user and record the channel-gain variations of the $\lceil \ell/2 \rceil$ subcarriers. Let us denote the two long trajectories of channel-gains by $\mathrm{Tr}_t(v_u)$ and $\mathrm{Tr}_e(v_u)$, respectively. $\mathrm{Tr}_t(v_u)$ will be used in the training phase and $\mathrm{Tr}_e(v_u)$ will be used in the evaluation phase. In the
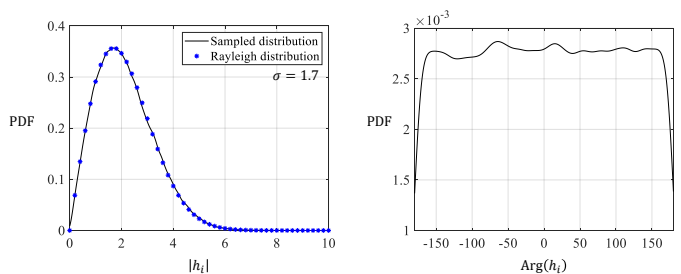
training phase, we randomly sample an initial point in $\mathrm{Tr}_t(v_u)$ and extract the channels gains of T consecutive slots starting from the initial point as the channel gains of one training epoch (one training epoch means one communication round to deliver a stream of K bits). Likewise, in the evaluation phase, we randomly sample an initial point in $\mathrm{Tr}_e(v_u)$ and extract the channels gains of T consecutive slots starting from the initial point as the channel gains of one evaluation epoch to assess the performance of a well-trained GBAF code model.

2) Sample statistics of the channel gains: Next, we analyze the sample statistics of the channel gains in $\mathrm{Tr}_t(v_u)$ and $\mathrm{Tr}_e(v_u)$. Two main results are as follows:

- Channel gains are correlated across subcarriers and interactions: across subcarriers, the fading is almost flat; across interactions, the amplitude and phase of the channel gains progressively increase or decrease.
- For each subcarrier, statistical properties of the channels are almost the same. Let us focus on one subcarrier and analyze the statistical properties of its amplitude and phase.
  Let $v_u = 1$ m/s and consider the i-th subcarrier, $i = 1, 2, ..., \lceil \ell/2 \rceil$. Fig. 9 presents the probability density function (PDF) of $|h_i|$ and $\mathrm{Arg}(h_i)$. As can be seen, the PDF of $|h_i|$ can be fitted by a Rayleigh distribution with $\sigma = 1.7$. The PDF of $\mathrm{Arg}(h_i)$, on the other hand, is approximately a uniform distribution. This indicates that the channel coefficients generated by QuaDRiGa for a single subcarrier can be viewed as Rayleigh fading.

3) Adapting GBAF code to fading channels: To apply GBAF code in fading channels, we assume that the channel gains are perfectly known to both nodes A (mobile user) and B (gNB). The received symbols at nodes A and B in fading channels can be written as

$$\boldsymbol{y} = \boldsymbol{h} \odot \boldsymbol{c} + \boldsymbol{n}, \tag{15}$$

$$\widetilde{\boldsymbol{y}} = \widetilde{\boldsymbol{h}} \odot \widetilde{\boldsymbol{c}} + \widetilde{\boldsymbol{n}} = \widetilde{\boldsymbol{h}} \odot \boldsymbol{h} \odot \boldsymbol{c} + \widetilde{\boldsymbol{h}} \odot \boldsymbol{n} + \widetilde{\boldsymbol{n}}, \tag{16}$$

where $\odot$ denotes the element-wise product; $\boldsymbol{h}$ and $\widetilde{\boldsymbol{h}}$ are the feedforward and feedback channel gains, respectively. In the case of reciprocal channels, we have $\boldsymbol{h} = \widetilde{\boldsymbol{h}}$.

Given the knowledge of $\boldsymbol{h}$, nodes A and B transform (15) and (16) to

$$\boldsymbol{y} \odot \frac{1}{\boldsymbol{h}} = \boldsymbol{c} + \boldsymbol{n} \odot \frac{1}{\boldsymbol{h}}, \tag{17}$$

$$\widetilde{\boldsymbol{y}} \odot \frac{1}{\boldsymbol{h}} \odot \frac{1}{\boldsymbol{h}} = \boldsymbol{c} + \boldsymbol{n} \odot \frac{1}{\boldsymbol{h}} + \widetilde{\boldsymbol{n}} \odot \frac{1}{\boldsymbol{h}} \odot \frac{1}{\boldsymbol{h}}. \tag{18}$$
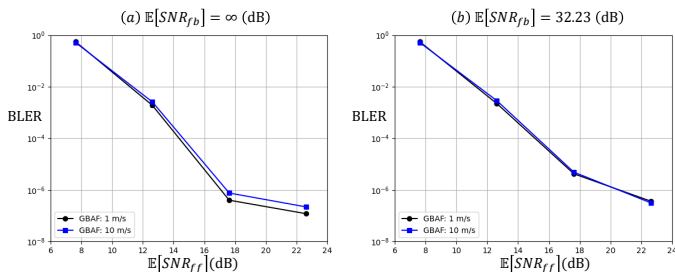
Fig. 10: Performance of GBAF code in fading channels benchmarked against DeepCode.

In so doing, the fading coefficients are transformed into the noise terms – the architecture of GBAF code can be used with the only difference being the non-AWGN noise.

4) Performance evaluation: Under the above setup, this subsection evaluates the performance of GBAF codes in fading channels. In the simulations, we generate the channel-gain trajectories using two user speeds $v_u = 1$ m/s and 10 m/s. The feedback SNR depends on both fading coefficients and noise power. When the feedback channel is noiseless (noise power is 0), the feedback SNR is simply $SNR_{fb} = \infty$ dB. When the feedback channel is noisy, we fix the average feedback SNR to $\mathbb{E}[SNR_{fb}] = 32.23$ dB (in which case the noise power is the same as that of the AWGN channel case with 20 dB noisy feedback).

Fig. 10 presents the BLER of GBAF code, where the feedback channel is noiseless in (a) and noisy in (b). Note that we do not simulate any prior works as benchmarks because they are designed exclusively for the unit-time delay case and do not fit into the considered NR-CDL model. As shown in Fig. 10, although faster mobile speed leads to faster-changing channel gains, GBAF code is robust to the mobile speed. The BLER performance only degrades slightly when the mobile speed increases from 1 m/s to 10 m/s.

V. CONCLUSION

In this work, we have introduced the novel GBAF code design, empowered by the sequence-to-sequence encoding DNN architecture, particularly the transformer architecture, to generate parity bits, by incorporating a feedback mechanism. Beyond introducing a generic code design framework, unlike the existing solutions described through the employed DNN architecture, the proposed framework addresses several limitations of the existing DNN-based coding approaches, and makes DNN-based feedback codes more applicable for next generation networks. In particular, our architecture is not limited to a fixed code rate, and achieves a significantly lower overhead thanks to its block structure. Finally, in addition to these operational advantages, GBAF codes significantly outperforms existing solutions, especially in the noiseless feedback scenario. This will be particularly attractive for applications where the feedback link is from a base station to a user equipment, and hence, can be assumed to achieve relatively high SNR values. We also showed that GBAF codes can be robust to channel fading, making them a promising alternative for mobile channels with relatively good average channel conditions.

APPENDIX A
IMPLEMENTATION DETAILS

The pre-processing unit of the encoder network can be operated under four different mode based on the enabled/disabled feedback mechanisms and the way the available information are aggregated. In Algorithm 3, we illustrate pre-processing mechanism under each mode with different color. In our implementation, we prefer the third option illustrated with blue. Here, we also note that in our implementation we store the original bits b in the knowledge vector in the BPSK modulated form, i.e., $\bar{b} = 2b - 1$. In overall, with enable/disable option for the belief network the GBAF code design can be operated under 8 different modes as highlighted in Algorithm 3.

APPENDIX B
FEATURE EXTRACTORS

The feature extractor presented in the main body of this paper is selected from a bunch of different designs. In this appendix, we discuss these designs and explain how the feature extractor is chosen.
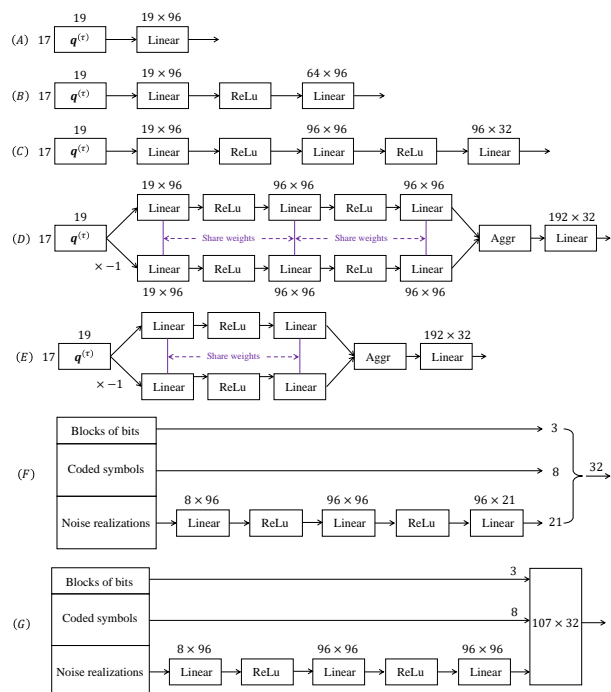
A. Various designs of the feature extractor

Fig. 11: Seven designs of the feature extractor.

We propose seven different designs of feature extractor for GBAF code, the architectures of which are summarized in Fig. 11. Note that variations of these designs can be obtained by changing the number of neurons in each layer or the activation functions.

Algorithm 3 Pre-processing unit for Parity Network: $S_{parity}()$

1: **if** Belief $b^{(\tau)}$ is available **then**
2:     **if** Feedback only is **True then**
3:         $q_i^{(\tau)} = [b_{((i-1)*m+1:i*m)}, \tilde{b}_i^{(\tau)}, \tilde{y}_i^{(1)}, \ldots, \tilde{y}_i^{(\tau-1)}])$
4:     **else if** Noise only is **True then**
5:         $q_i^{(\tau)} = [b_{((i-1)*m+1:i*m)}, \tilde{b}_i^{(\tau)}, \tilde{y}_i^{(1)} - c_i^{(1)}, \ldots, \tilde{y}_i^{(\tau-1)} - c_i^{(\tau-1)}])$
6:     **else if** Disentangle is **True then**
7:         $q_i^{(\tau)} = [b_{((i-1)*m+1:i*m)}, \tilde{b}_i^{(\tau)}, c_i^{(1)}, \ldots, c_i^{(\tau-1)}, \tilde{y}_i^{(1)} - c_i^{(1)}, \ldots, \tilde{y}_i^{(\tau-1)} - c_i^{(\tau-1)}]$
8:     **else**
9:         $q_i^{(\tau)} = [b_{((i-1)*m+1:i*m)}, \tilde{b}_i^{(\tau)}, c_i^{(1)}, \ldots, c_i^{(\tau-1)}, \tilde{y}_i^{(1)}, \ldots, \tilde{y}_i^{(\tau-1)}]$
10: **else**
11:     **if** Feedback only is **True then**
12:         $q_i^{(\tau)} = [b_{((i-1)*m+1:i*m)}, \tilde{y}_i^{(1)}, \ldots, \tilde{y}_i^{(\tau-1)}])$
13:     **else if** Noise only is **True then**
14:         $q_i^{(\tau)} = [b_{((i-1)*m+1:i*m)}, \tilde{y}_i^{(1)} - c_i^{(1)}, \ldots, \tilde{y}_i^{(\tau-1)} - c_i^{(\tau-1)}])$
15:     **else if** Disentangle is **True then**
16:         $q_i^{(\tau)} = [b_{((i-1)*m+1:i*m)}, c_i^{(1)}, \ldots, c_i^{(\tau-1)}, \tilde{y}_i^{(1)} - c_i^{(1)}, \ldots, \tilde{y}_i^{(\tau-1)} - c_i^{(\tau-1)}]$
17:     **else**
18:         $q_i^{(\tau)} = [b_{((i-1)*m+1:i*m)}, c_i^{(1)}, \ldots, c_i^{(\tau-1)}, \tilde{y}_i^{(1)}, \ldots, \tilde{y}_i^{(\tau-1)}]$

- Design A is simply a linear layer. This is the feature extractor used in the original design of transformer [16].
- Design B consists of two linear layers with a ReLU activation function in between. The output of ReLU is 0 when the input is negative. Therefore, it can be used for truncation: whenever the DNN wants to truncate a large output of a neuron (large in amplitude), say z, it can simply multiply z by a weight -1 (when $z > 0$) or 1 (when $z < 0$) and then feed $-z$ or z into ReLU, yielding ReLU$(-|z|) = 0$.

  With design B, we hope the ReLU non-linearity can truncate noise realizations with large amplitude, mimicking the modulo operation used in modulo-SK [9]. It is worth noting that ReLU can be replaced by GeLU, which will be discussed later.

- Design C is an extension of design B, where we use three linear layers with two ReLU activation functions in between. A single truncation layer in design B can only truncate either positive or negative noise realizations when the weights and bias of the linear layers are fixed. This motivates us to add an additional ReLU truncation in Design C such that one ReLU can truncate positive noise realizations and the other can truncate negative noise realizations.

  Design C is the final design we choose as the default feature extractor for GBAF code.

- In Design D, we use two parallel noise suppression flows and each flow is the same as design C. In particular, 1) for the second flow, the noise realization part is multiplied by -1 as the input; 2) the two parallel flows share the same weights. In doing so, we ask the DNN to tackle the positive noise realization in one flow and the negative noise realization in the other. After processing by two parallel branches, the resulting features are aggregated and transformed by a linear layer to obtain the output.
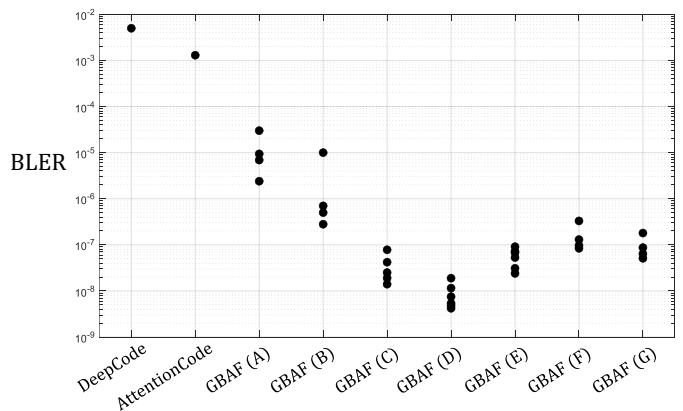

Fig. 12: BLER performance of various designs. The system setup is $K = 51$, $N = 153$, $m = 3$, $\ell = 17$, the feedforward SNR is fixed to -1 dB, and the feedback is noiseless. One black point stands for one independent simulations.

- Design E is a simplification of design D, where each of the two parallel flows is chosen to be design B, as opposed to design C.
- Designs F and G are extensions of design C. Specifically, instead of processing the whole feature matrix, we only process the noise realizations by design C. After feature extraction, we aggregate the extracted features with bits and coded symbols to obtain the final features to be fed into the attention network. The difference between designs F and G lies in the last aggregation step: design F simple aggregates the features, while the aggregation in design G is followed by a linear transformation.

B. Architecture selection

This section compares the seven designs in Fig. 11 considering the noiseless feedback system. In particular, we fix
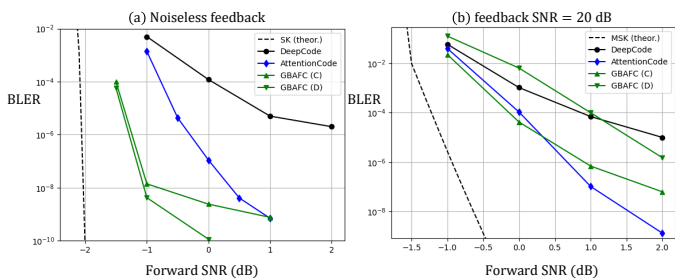
Fig. 13: BLER versus feedforward SNR for various feedback codes. The system setup is $K = 51$, $N = 153$, $m = 3$, $\ell = 17$.

Fig. 14: Performance comparison between ReLU and GeLU when used in feature extractor (design C) of GBAF code. The system setup is $K = 51$, $N = 153$, $m = 3$, $\ell = 17$. (a) noiseless feedback, (b) noisy feedback.

Fig. 15: Performance comparison between ReLU and GeLU when used in feature extractor (design C) of GBAF code. The system setup is $K = 51$, $N = 153$, $m = 3$, $\ell = 17$. GBAF code uses both feature extractor and belief network. (a) noiseless feedback, (b) 20 dB noisy feedback.

the feedforward SNR to -1 dB and simulate each design multiple times. The BLER performances of various designs are presented in Fig. 12. To showcase the impact of feature extractors, we do not incorporate belief network into GBAF code.

As can be seen from Fig. 12, with the seven designs of feature extractor, GBAF code can achieve a BLER as low as 5×10^{-9} at a feedforward SNR of -1 dB. As far as the BLER performance is concerned, the most preferable architectures are designs C and D. On the other hand, as far as the computational resource is concerned, design C is twice more efficient than design D. In the following, we will focus on designs C and D, and perform extensive simulations on a wide range of SNRs and compare their performances with other feedback codes.

Fig. 13 compares the BLER performance of different feedback codes with noiseless and noisy feedback. In particular, we plot the theoretical performances of the SK and modulo-SK schemes [2], [9] as benchmarks. In the noiseless feedback case, GBAF code with either design C or design D achieves significant performance gains over AttentionCode. To attain a BLER of 10^{-8}, GBAF code with designs C or D is about 1.2 dB better than AttentionCode. In the noisy feedback case (20 dB feedback SNR), the performance of GBAF code with design D deteriorates for a large margin. On the other hand, GBAF code with design C still performs well.

Based on the simulation results in this section, we choose design C as the default feature extractor for GBAF code. In the noiseless feedback case, design C is slightly worse than design D and outperforms all other designs. In the noisy feedback case, design C is much better than design D. In addition, design C is twice more efficient than design D as far as the required computational resource is concerned.

C. ReLU versus GeLU

In the machine learning community, GeLU activation function is widely considered as a better alternative to ReLU, thanks to its differentiability at $x = 0$ for an input x. In our simulations, we find that the choice of the activation function is crucial to the performance of GBAF codes, and GeLU is better than ReLU only when an ultra-low BLER is to be achieved. In this section, we perform extensive simulations to compare the performance of ReLU and GeLU when used in the feature extractor (design C) of the GBAF code.
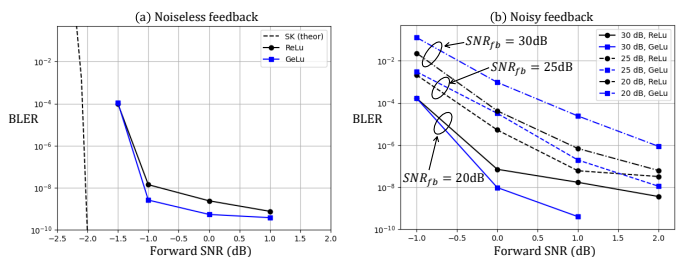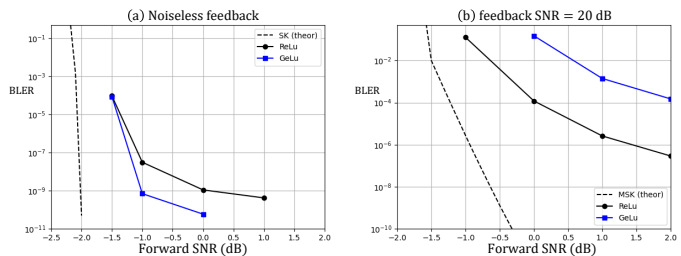
Fig. 14(a) compares the performance of ReLU and GeLU when used in the feature extractor, where the feedback is assumed to be noiseless. As can be seen, GeLU is beneficial to the BLER performance. Compared with ReLU, the performance of GBAF code is improved by 5 times at a feedforward SNR of -1 dB when GeLU is used as the activation function.

Fig. 14(b) compares the performance of ReLU and GeLU in the noisy feedback setup, where we progressively decrease the feedback SNR from 30 dB to 20 dB. As shown, when both the feedforward and feedback SNRs are large (e.g., $SNR_{ff} > 1$ dB and $SNR_{fb} \geq 25$ dB), GeLU exhibits better performance than ReLU. In contrast, when the feedforward and feedback SNRs decrease, ReLU performs better. As a conclusion, if an ultra-low BLER (e.g., BLER lower than 10^{-7}) is to be achieved, GeLU is a better choice than ReLU.

To confirm the above results, we perform additional simulations by taking the belief network into account. Specifically, we consider GBAF code with both feature extractor (design C) and belief network. When ReLU and GeLU are used as the activation function of the feature extractor, Fig. 15 compares the performance achieved by GBAF code. When the feedback channel is noiseless, GeLU yields much lower BLER than ReLU. When the feedback channel is noisy (20 dB), on the other hand, ReLU is better choice than GeLU. This confirms our conclusions drawn from Fig. 14.

To conclude, the default feature extractor for GBAF code is set to be design C with the ReLU activation function. An alternative to ReLU is GeLU when the target BLER is ultra low (e.g., lower than 10^{-7}).

REFERENCES

[1] C. Shannon, "The zero error capacity of a noisy channel," *IRE Trans.Inf. Theory*, vol. 2, no. 3, pp. 8–19, 1956.

[2] J. Schalkwijk and T. Kailath, "A coding scheme for additive noise channels with feedback i: No bandwidth constraint," *IEEE Trans. Inf. Theory*, vol. 12, no. 2, pp. 172–182, 1966.

[3] J. Schalkwijk, "A coding scheme for additive noise channels with feedback ii: Band-limited signals," *IEEE Trans. Inf Theory*, vol. 12, no. 2, pp. 183–189, 1966.

[4] K. Zigangirov, "Upper bounds for the error probability for channels with feedback," *Problemy Peredachi Informatsii*, vol. 6, no. 2, pp. 87–92, 1970.

[5] N. C. Martins and T. Weissman, "Coding for additive white noise channels with feedback corrupted by quantization or bounded noise," *IEEE Trans. Inf. Theory*, vol. 54, no. 9, pp. 4274–4282, 2008.

[6] R. G. Gallager and B. Nakiboğlu, "Variations on a theme by Schalkwijk and Kailath," *IEEE Trans. on Inf. Theory*, vol. 56, no. 1, pp. 6–17, 2009.

[7] Z. Chance and D. J. Love, "Concatenated coding for the AWGN channel with noisy feedback," *IEEE Trans. Inf. Theory*, vol. 57, no. 10, pp. 6633–6649, 2011.

[8] Y.-H. Kim, A. Lapidoth, and T. Weissman, "The gaussian channel with noisy feedback," in *IEEE ISIT*, 2007, pp. 1416–1420.

[9] A. Ben-Yishai and O. Shayevitz, "Interactive schemes for the awgn channel with noisy feedback," *IEEE Trans. Inf. Theory*, vol. 63, no. 4, pp. 2409–2427, 2017.

[10] H. Kim, Y. Jiang, S. Kannan, S. Oh, and P. Viswanath, "Deepcode: Feedback codes via deep learning," *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 194–206, 2020.

[11] A. R. Safavi, A. G. Perotti, B. M. Popovic, M. B. Mashhadi, and D. Gunduz, "Deep extended feedback codes," *ITU Journal on Future and Evolving Technologies - Wireless communication systems in beyond 5G era*, vol. 2, no. 6, pp. 33–41, 2021.

[12] M. B. Mashhadi, D. Gündüz, A. Perotti, and B. M. Popovic, "DRF codes: Deep SNR-robust feedback codes," *CoRR*, vol. abs/2112.11789, 2021. [Online]. Available: https://arxiv.org/abs/2112.11789

[13] Y. Shao, E. Ozfatura, A. Perotti, B. Popovic, and D. Gunduz, "Attention-code: Ultra-reliable feedback codes for short-packet communications," *arXiv:2205.14955*, 2022.

[14] M. Shirvanimoghaddam, M. S. Mohammadi, R. Abbas, A. Minja *et al.*, "Short block-length codes for ultra-reliable low latency communications," *IEEE Commun. Magazine*, vol. 57, no. 2, pp. 130–137, 2018.

[15] Y. Shao, S. C. Liew, and J. Liang, "Sporadic ultra-time-critical crowd messaging in V2X," *IEEE Transactions on Communications*, vol. 69, no. 2, pp. 817–830, 2020.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Proc. Systems*, vol. 30, 2017.

[17] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[18] Y. Wang, Z. Gao, D. Zheng, S. Chen, D. Gunduz, and H. V. Poor, "Transformer-empowered 6G intelligent networks: From massive mimo processing to semantic communication," *IEEE Wireless Communications Magazine*, 2022.

[19] Y. Shao and D. Gunduz, "Semantic communications with discrete-time analog transmission: A PAPR perspective," *arXiv:2208.08342*, 2022.

[20] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *CoRR*, vol. abs/1409.1259, 2014.

[21] H. Sak, A. W. Senior, and F. Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," *CoRR*, vol. abs/1402.1128, 2014. [Online]. Available: http://arxiv.org/abs/1402.1128

[22] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[23] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Advances in Neural Information Processing Systems*, vol. 26, 2013.

[24] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," 2013.

[25] D. Hendrycks and K. Gimpel, "Gaussian error linear units (GELUs)," *arXiv preprint arXiv:1606.08415*, 2016.

[26] M. Geva, R. Schuster, J. Berant, and O. Levy, "Transformer feed-forward layers are key-value memories," *CoRR*, vol. abs/2012.14913, 2020. [Online]. Available: https://arxiv.org/abs/2012.14913

[27] R. Xiong, Y. Yang, D. He, K. Zheng, S. Zheng, C. Xing, H. Zhang, Y. Lan, L. Wang, and T. Liu, "On layer normalization in the transformer architecture," *CoRR*, vol. abs/2002.04745, 2020. [Online]. Available: https://arxiv.org/abs/2002.04745

[28] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," *CoRR*, vol. abs/2106.04554, 2021. [Online]. Available: https://arxiv.org/abs/2106.04554

[29] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Bkg6RiCqY7

[30] Huawei-HiSilicon, "Performance evaluation of LDPC codes for NR eMBB data," in *R1-1713740, 3GPP RAN1 meeting 90*, August 2017. [Online]. Available: https://www.3gpp.org/dynareport?code=TDocExMtg--R1-90--17073.htm

[31] N. Kitaev, L. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," *CoRR*, vol. abs/2001.04451, 2020.

[32] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," *CoRR*, vol. abs/2006.04768, 2020. [Online]. Available: https://arxiv.org/abs/2006.04768

[33] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlós, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, D. Belanger, L. J. Colwell, and A. Weller, "Rethinking attention with performers," *CoRR*, vol. abs/2009.14794, 2020.

[34] S. Jaeckel, L. Raschkowski, K. Börner, and L. Thiele, "Quadriga: A 3-D multi-cell channel model with time evolution for enabling virtual field trials," *IEEE Trans. Antennas and Propagation*, vol. 62, no. 6, pp. 3242–3256, 2014.