Transformer-Based Decoding in Concatenated Coding Schemes Under Synchronization Errors

Julian Streit

School of Computation, Information and Technology, Technical University of Munich
Munich Center for Machine Learning
Email: julian.streit@tum.de

***Abstract*—We consider the reconstruction of a codeword from multiple noisy copies that are independently corrupted by insertions, deletions, and substitutions. This problem arises, for example, in DNA data storage. A common code construction uses a concatenated coding scheme that combines an outer linear block code with an inner code, which can be either a nonlinear marker code or a convolutional code. Outer decoding is done with Belief Propagation, and inner decoding is done with the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm. However, the BCJR algorithm scales exponentially with the number of noisy copies, which makes it infeasible to reconstruct a codeword from more than about four copies. In this work, we introduce *BCJRFormer*, a transformer-based neural inner decoder. *BCJRFormer* achieves error rates comparable to the BCJR algorithm for binary and quaternary single-message transmissions of marker codes. Importantly, *BCJRFormer* scales quadratically with the number of noisy copies. This property makes *BCJRFormer* well-suited for DNA data storage, where multiple reads of the same DNA strand occur. To lower error rates, we replace the Belief Propagation outer decoder with a transformer-based decoder. Together, these modifications yield an efficient and performant end-to-end transformer-based pipeline for decoding multiple noisy copies affected by insertion, deletion, and substitution errors. Additionally, we propose a novel cross-attending transformer architecture called *ConvBCJRFormer*. This architecture extends *BCJRFormer* to decode transmissions of convolutional codewords, serving as an initial step toward joint inner and outer decoding for more general linear code classes.**

I. Introduction

DNA data storage is an emerging storage medium that encodes binary data into DNA sequences for high-density, long-term storage. However, writing, storing, and reading DNA are error-prone processes that produce multiple noisy reads of the same sequence [1]. Errors can be categorized into two categories: (1) substitutions and erasures, and (2) insertions and deletions. While optimal linear codes are known for erasures and substitutions, no similar guarantees exist for insertions and deletions.

A common approach to address insertion, deletion, and substitution errors is to use a concatenated coding scheme that uses an inner and an outer encoder/decoder pair. The inner code is often nonlinear — using, for example, marker codes [2] or watermark codes [3] — and is typically decoded with the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm. Under the assumption of perfect channel state information, the BCJR algorithm achieves maximum a posteriori decoding. The outer code is a linear block code, such as a low-density parity-check (LDPC) code [4] or a polar code [5], and is decoded using Belief Propagation. Belief Propagation achieves maximum a posteriori decoding when the code's Tanner graph has no cycles [6].

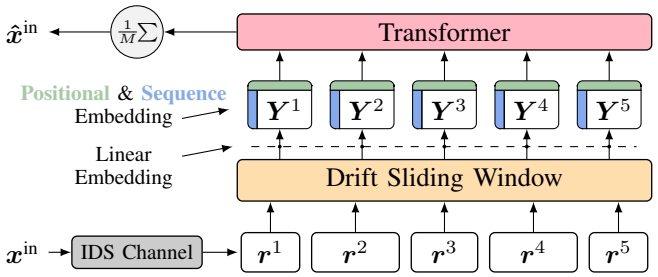


Fig. 1: Overview of *BCJRFormer* for jointly decoding marker sequences over the IDS channel.

The computational complexity of the BCJR algorithm scales exponentially with the number of input sequences, which makes joint decoding over multiple noisy copies infeasible. To address this limitation, we propose *BCJRFormer*, a transformer-based architecture for inner decoding of marker codes. The architecture achieves error rates close to those of joint decoding with the BCJR algorithm while scaling only quadratically with the number of noisy copies.

To further reduce bit error rates, we replace the outer Belief Propagation algorithm with the Error Correcting Code Transformer (ECCT) proposed in [7]. The authors show that their ECCT decoder outperforms both Belief Propagation and other neural decoders on binary-input symmetric-output channels. By adapting the input to ECCT, we propose a two-step decoding approach for LDPC codes concatenated with a marker code. This approach reduces bit error rates and improves the efficiency of decoding for correcting insertion, deletion, and substitution errors compared to traditional Belief Propagation and BCJR decoding.

Separating inner and outer decoding is necessary because incorporating the state-space of general linear codes into the BCJR algorithm is computationally infeasible. Convolutional codes are an exception because they exhibit a sparse diagonal structure [8]. As an initial step toward joint inner and outer decoding of more general linear code classes, we extend

BCJRFormer to convolutional codes by using a cross-attention mechanism that incorporates their state structure. We demonstrate that this decoder, named *ConvBCJRFormer*, achieves error rates only slightly worse than BCJR decoding while offering a generalizable mechanism to incorporate linear code information.

We summarize our contributions as follows:[1]

- We introduce a sliding window input representation derived from the BCJR algorithm. This approach yields *BCJRFormer*, a transformer model that achieves error rates comparable to those of the BCJR algorithm in single-sequence transmissions.
- We extend *BCJRFormer* to jointly decode multiple copies of transmitted sequences and demonstrate that the resulting decoder achieves error rates similar to the BCJR algorithm while operating with only quadratic complexity.
- We demonstrate that ECCT can serve as an outer decoder in concatenated coding schemes, achieving lower error rates than Belief Propagation.
- By combining *BCJRFormer* as the inner decoder with ECCT as the outer decoder (see Figure 4), we propose a transformer-based pipeline for end-to-end decoding of concatenated codewords over the insertion, deletion, and substitution (IDS) channel that outperforms approaches based on BCJR or Belief Propagation decoders.
- We introduce *ConvBCJRFormer*, a cross-attending transformer architecture that extends *BCJRFormer* to decode convolutional codes as an initial step toward transformer-based joint inner and outer decoding of linear codeword transmissions over the IDS channel.

II. Related Work

In DNA data storage, multiple noisy reads of the same sequence are often clustered and then aligned to yield a candidate sequence for error correction [9]–[12]. Because the BCJR algorithm becomes infeasible for larger cluster sizes due to its exponential complexity, the article [13] analyzes decoding convolutional codes by applying the BCJR algorithm to each read independently and then multiplying the resulting a posteriori probabilities. The paper [14] builds on this approach by proposing Trellis BMA — an algorithm that combines separate trellis calculations with bitwise majority alignment.

The article [10] proposes DNAformer, an end-to-end retrieval solution for DNA data storage that also uses a transformer-based architecture to reconstruct data from multiple noisy reads. Their work focuses on the challenge of imperfect clustering, which can result in reads coming from different sequences. A recent paper [11] formulates the problem of reconstructing an uncoded DNA sequence from multiple noisy copies as a next token prediction task and evaluates decoder-only transformers for this purpose. Although the tasks differ, we note that their transformers have millions of parameters, whereas our *BCJRFormer* has fewer than one million.

[1]Our code is available at https://github.com/streit-julian/BCJRFormer.

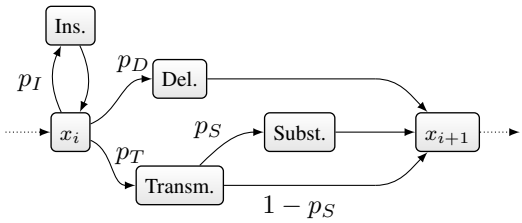


Fig. 2: State transitions for a codeword symbol x_i transmitted through the IDS channel.

Several neural decoders have been proposed to replace iterative decoding algorithms — such as BCJR or Belief Propagation — across various channel models [15]–[20]. We distinguish between model-based and model-free architectures [21].

In the paper [22], the authors propose two model-based recurrent neural network (RNN) architectures — FBNet and FBGRU — to decode insertion, deletion, and substitution errors in marker codes. Their models outperform the BCJR algorithm when the channel state information is imperfect. The input for their models is derived from the BCJR algorithm, much like how we design the input for our *BCJRFormer*.

The authors of the article [23] propose a bidirectional gated recurrent unit (BiGRU) decoder to address vanishing and exploding gradient issues in RNNs. They use one BiGRU network for inner decoding of marker codes and a second BiGRU network for outer decoding of codewords encoded with an LDPC code or a convolutional code to correct deletion and substitution errors.

Model-free networks are characterized by more general neural architectures, but are often harder to design because the network must learn both the code structure and the reconstruction process. The authors of the paper [7] propose using the attention mechanism in transformer networks to embed linear code information directly into the model architecture. They introduce ECCT, which restricts the attention mechanism to attend only to those input bits which are related by parity-check equations. ECCT outperforms existing neural decoders on memoryless binary-input symmetric-output channels.

More recently, the article [24] proposes CrossMPT to improve the decoding performance of ECCT. CrossMPT is also transformer-based and uses two masked cross-attention modules to handle channel noise and syndrome, imitating the check and variable node updates of the Belief Propagation algorithm. We employ a similar cross-attention mechanism to decode inner convolutional codes; however, we derive the mask from the code's generator matrix rather than from the parity-check matrix, which is used in CrossMPT's approach.

III. Background

This section presents an overview of the key concepts relevant to our work. We first introduce our channel model and describe concatenated coding schemes, with a focus on the BCJR algorithm that is used to derive the input for both *BCJRFormer* and *ConvBCJRFormer*. We then discuss the

transformer architecture and the original formulation of the ECCT decoder for binary-input symmetric-output channels.

A. Channel Model

We consider transmission via the insertion, deletion, and substitution channel over a finite field $\mathbb{F}_q$ of order $q = 2^p$, where p is a positive integer. The IDS channel is represented as a state machine, as we show in Figure 2. Each symbol x_i in a sequence $\boldsymbol{x} \in \mathbb{F}_q^n$ enters the state machine independently. The symbol is deleted with probability p_D. With insertion probability p_I, the channel outputs a random symbol and resets its state. With transmission probability $p_T = 1 - p_I - p_D$, the symbol is transmitted. In this case, the symbol is substituted with probability p_S by a different symbol chosen uniformly at random [3].

B. Concatenated Schemes and Linear Codes

Concatenated coding schemes consist of an inner encoder/decoder pair and an outer encoder/decoder pair. A message $\boldsymbol{m} \in \mathbb{F}_q^k$ is first transformed by the outer encoder into an outer codeword $\boldsymbol{x}^{\text{out}} \in \mathbb{F}_q^{n_{\text{out}}}$. This outer codeword is then encoded by the inner encoder to produce the inner codeword $\boldsymbol{x}^{\text{in}} \in \mathbb{F}_q^{n_{\text{in}}}$. After transmission, the inner decoder synchronizes the received sequence $\boldsymbol{r} \in \mathbb{F}_q^{n_{\text{rec}}}$ and computes log-likelihood values that serve as soft information for the outer decoder. The outer decoder then uses this soft information, together with the structure of the outer code, to correct any remaining substitution errors.

The outer code is typically a linear block code. A (k, n_{out}) linear block code is defined by its generator matrix $\boldsymbol{G} \in \mathbb{F}_q^{k \times n_{\text{out}}}$. Multiplying the message $\boldsymbol{m}$ by the generator matrix produces the codeword $\boldsymbol{x}^{\text{out}} = \boldsymbol{m}\boldsymbol{G}$. The parity-check matrix $\boldsymbol{H} \in \mathbb{F}_q^{(n_{\text{out}}-k) \times n_{\text{out}}}$ is defined so that $\boldsymbol{H}\boldsymbol{x}^{\text{out}} = \boldsymbol{0}$ holds for every codeword. The parity-check matrix can be represented by a Tanner graph, a bipartite graph in which the rows of the parity-check matrix serve as check nodes and the columns as variable nodes [6]. A check node and a variable node are connected by a weighted edge if the corresponding entry of the parity-check matrix is non-zero, with the edge weight given by that entry.

The Belief Propagation algorithm operates on the Tanner graph by iteratively passing soft information between check and variable nodes to reconstruct the original message. Belief Propagation yields maximum a posteriori predictions of the codeword if the Tanner graph is cycle-free. If cycles are present, Belief Propagation still provides effective approximations. In this work, we focus on LDPC outer codes [25]. LDPC codes are a class of linear codes designed for efficient decoding via message passing.

C. Convolutional Codes

A $(n_{\text{c}}, k_{\text{c}}, m)$ convolutional code with rate $k_{\text{c}}/n_{\text{c}}$ and memory m is described by k_{c} generator polynomials $g = [g_1, g_2, \dots g_{k_{\text{c}}}]$, where each $g_l \in \mathbb{F}_q[x]^{n_{\text{c}}}$ is a vector of n_{c} polynomials with degree at most m. For brevity, we express the polynomials in octal notation. For example, we denote a pair of generator polynomials as $[5, 7]_8$, where each digit represents the binary coefficients of a polynomial in octal form. For example, $[5, 7]_8$ represents the polynomials 101 and 111.

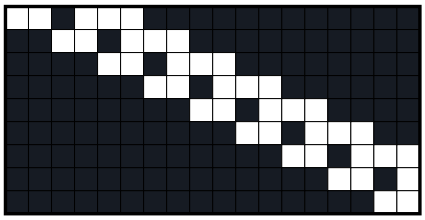


Fig. 3: Generator matrix $G \in \{0, 1\}^{9 \times 18}$ for the $(2, 1, 2)$ convolutional code with generator polynomials $[5, 7]_8$, used for an outer codeword with $n_{\text{out}} = 7$. White entries indicate unmasked positions.

Each generator polynomial g_l is applied to one of the k_{c} input symbols. At time i, the encoder uses a sliding window of $m+1$ inputs — the current input and the m preceding inputs — to produce n_{c} output symbols. This process can be viewed as a Markov chain: the encoder's state is given by the contents of its memory, so transitions from time i to time $i+1$ depend only on the current state and the new input.

For fixed transmission lengths, the decoder benefits from knowing the final memory state. To provide this information, we zero-terminate the code by appending the zero vector $\boldsymbol{0}^m$ to the codeword. Consequently, given an outer codeword with length n_{out}, the inner codeword has length $n_{\text{in}} = \frac{n_{\text{c}}}{k_{\text{c}}}(n_{\text{out}} + m)$.

An inner convolutional code can also be interpreted as a linear block code. In this perspective, the generator polynomials define a structured generator matrix G that maps an input vector (augmented by the m zero-termination symbols) to the codeword. The structure of G is typically banded, with each row being a shifted version of the generator polynomials. Figure 3 shows the generator matrix for a rate $1/2$ convolutional code with generator polynomials $[5, 7]_8$ as an example.

After encoding, we add a pseudorandom offset $\boldsymbol{o} \in \mathbb{F}_2^{n_{\text{in}}}$ to the convolutional codeword. This offset, which is known to the decoder, improves error rates by mitigating the cyclic structure of convolutional codes [26].

D. Inner Code Construction and BCJR

Inner marker codes insert a fixed marker sequence $\boldsymbol{s}_{\text{m}} \in \mathbb{F}_q^{n_{\text{m}}}$ at a fixed interval N_{m} between the symbols of the outer codeword. The resulting inner codeword has length $n_{\text{in}} = n_{\text{out}} + n_{\text{m}}\lfloor \frac{n_{\text{out}}}{N_{\text{m}}} \rfloor$. These markers enable the inner decoder to synchronize the received sequence $\boldsymbol{r}$.

For each inner codeword symbol x_i^{in} (where $1 \le i \le n_{\text{in}}$), the inner decoder estimates the a posteriori probabilities

$$P(x_i^{\text{in}} = \xi \,|\, \boldsymbol{r}) \propto P(x_i^{\text{in}} = \xi, \boldsymbol{r}), \tag{1}$$

where $\xi \in \mathbb{F}_q = \{0, \dots, q-1\}$. The BCJR algorithm is a forward-backward method that leverages the IDS channel's Markov property to compute these probabilities. Assuming that the symbols of the outer codeword are independent and identically distributed, the BCJR algorithm yields exact results.

In this work, we derive the algorithm specifically for marker codes.

To elicit the IDS channel's Markov property, we follow the paper [3] and introduce a latent drift variable D_i, defined as the difference between the number of insertions and deletions that occur before transmitting the i-th symbol. Transmitting the i-th symbol results in a state transition from state D_i to D_{i+1}. Each state transition emits between 0 and $I_{\max}+1$ symbols, where $I_{\max}$ is a parameter that limits the number of consecutive insertions per symbol to reduce computational complexity. Consequently, if $D_i = d$, then D_{i+1} can take any value in the set $\{d-1,\, d,\, d+1,\, \ldots,\, d+I_{\max}\}$. Using the Markov property, we rewrite the joint probabilities as

$$P(x_i^{\text{in}} = \xi,\, \boldsymbol{r}) = \sum_{d} \sum_{d'=d-1}^{d+I_{\max}} P(x_i^{\text{in}} = \xi,\, \boldsymbol{r}, d, d'),$$

where the first summation is over all possible realizations d of the drift variable D_{i-1}. We factorize the term in the second summation as follows:

$$\begin{aligned} P(x_i^{\text{in}} = \xi,\, \boldsymbol{r}, d, d') =& P(\boldsymbol{r}_1^{i+d-1}, d) \\ & \cdot P(x_i^{\text{in}} = \xi,\, \boldsymbol{r}_{i+d}^{i+d'}, d' \mid d) \\ & \cdot P(\boldsymbol{r}_{i+d'+1}^{n_{\text{in}}} \mid d'), \end{aligned} \tag{2}$$

where we denote by $\boldsymbol{r}_a^b$ the sequence $r_a,\, r_{a+1},\, \ldots,\, r_b$ of the vector $\boldsymbol{r}$. We abbreviate the three factors in Equation (2) in order of appearance as $\alpha_{i-1}(d)$, $\gamma_i(d,\, d')$, and $\beta_i(d')$. We deduce BCJR's forward (α_i) and backward (β_i) recursions as

$$\begin{aligned} \alpha_i(d') =& \frac{p_I}{q}\alpha_i(d'-1) + p_D\alpha_{i-1}(d'+1) \\ & + p_T\alpha_{i-1}(d') \sum_{\xi\in\mathbb{F}_q} P(x_i^{\text{in}} = \xi) F(\xi, r_{i+d'}), \end{aligned}$$

and

$$\begin{aligned} \beta_i(d) =& \frac{p_I}{q}\beta_i(d+1) + p_D\beta_{i+1}(d+1) \\ & + p_T\beta_{i+1}(d) \sum_{\xi\in\mathbb{F}_q} P(x_i^{\text{in}} = \xi) F(\xi, r_{i+1+d}), \end{aligned}$$

where

$$F(\xi, r_i) = \begin{cases} \frac{p_S}{q-1} & \text{if } \xi \neq r_i, \\ 1-p_S & \text{else.} \end{cases}$$

Since the initial drift (0) and final drift ($n_{\text{rec}} - n_{\text{in}}$) are known, the initial conditions for both recursions are

$$\alpha_0(d') = \begin{cases} 1 & \text{if } d' = 0, \\ 0 & \text{else,} \end{cases}$$

and

$$\beta_{n_{\text{in}}}(d) = \begin{cases} 1 & \text{if } d = n_{\text{rec}} - n_{\text{in}}, \\ 0 & \text{else.} \end{cases}$$

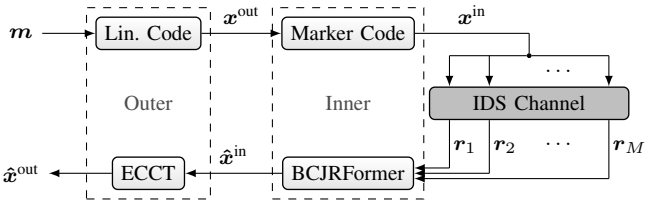


Fig. 4: Joint decoding of concatenated codes using *BCJRFormer* as the inner decoder and ECCT as the outer decoder.

By combining the forward and backward recursions with the branching metrics γ_i, we simplify the calculation of the joint probabilities in Equation (1) to obtain

$$\begin{aligned} P(x_i^{\text{in}} = \xi,\, \boldsymbol{r}) = \sum_{d} & \frac{p_I}{q}\alpha_i(d-1)\beta_i(d) \\ & + p_D\alpha_{i-1}(d+1)\beta_i(d) \\ & + p_T\alpha_{i-1}(d)\beta_i(d)F(\xi, r_{i+d}). \end{aligned}$$

For a more detailed discussion of the BCJR algorithm, we refer the reader to the works [27], [28]. The BCJR algorithm can also decode convolutional inner codes over the IDS channel by considering all combinations of convolutional and drift states [13].

E. Transformer and ECCT

The transformer architecture uses attention to capture dependencies across the input sequence [29]. Let d_{k} denote the model's embedding dimension. An attention layer is defined as

$$Att(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_{\text{k}}}}\right)\boldsymbol{V}, \tag{3}$$

where the query $\boldsymbol{Q}$, key $\boldsymbol{K}$, and value $\boldsymbol{V}$ matrices are learned from the input tokens. If all three matrices are derived from the same input sequence, this is called self-attention. Each attention layer consists of n_{h} attention heads. Each head performs attention on a reduced dimension of $d_{\text{k}}/n_{\text{h}}$. The outputs of all heads are concatenated and then passed through a final linear layer.

A transformer stacks multiple sequential layers. Each layer comprises an attention block followed by a feed-forward block. In the attention block, the transformer first normalizes its input, then applies multi-head attention, and finally adds a residual connection. After processing all layers, the transformer projects the final output to a task-specific dimension.

The ECCT was proposed for decoding transmissions over a binary-input symmetric-output channel, a type of memory-free channels that introduces substitution errors independently of the binary input symbol [7].

ECCT incorporates linear code information by masking unrelated bit positions in its self-attention module. The model's input is constructed to be independent of the transmitted codeword.

Before transmission over a binary-input symmetric-output channel, we modulate the codeword using binary phase-shift keying. In this modulation, each bit is first mapped to a bipolar value — specifically, 0 is mapped to 1 and 1 is mapped

to -1. Let $\boldsymbol{r}_{\text{bpsk}} \in [-1,1]^{n_{\text{out}}}$ denote the received sequence corresponding to this binary phase-shift keying-modulated codeword. The model's input is formed by concatenating the magnitude vector $|\boldsymbol{r}_{\text{bpsk}}| \in \mathbb{R}^{n_{\text{out}}}$ with the syndrome vector $\phi(\text{syn}(\text{bin}(\boldsymbol{r}_{\text{bpsk}}))) \in \{-1,1\}^{n_{\text{out}}-k}$. Here, we define the syndrome of a sequence as $\text{syn}(\boldsymbol{x}) = \boldsymbol{H}\boldsymbol{x} \in \mathbb{F}_2^{n_{\text{out}}-k}$, the bipolar mapping as $\phi(\boldsymbol{x}) = 1 - 2\boldsymbol{x}$, and the binarization as $\text{bin}(\boldsymbol{x}) = 0.5(1 - \text{sign}(\boldsymbol{x}))$.

By concatenating these two vectors, the model captures the channel's multiplicative noise independently of the transmitted codeword, which enables training using only the all-zero sequence [30].

IV. Method

In this section, we introduce *BCJRFormer*, a neural decoder for efficiently decoding multiple noisy copies of inner codewords encoded with marker codes. We also describe our modifications to the ECCT model for use as an outer decoder alongside *BCJRFormer*, which leads to lower error rates for transmissions over the IDS channel. We then propose the *ConvBCJRFormer* architecture, an extension of *BCJRFormer* to jointly synchronize and decode convolutional codes.

A. BCJRFormer

Our proposed *BCJRFormer* model, illustrated in Figure 1, is a decoder-only transformer. We focus on constructing an appropriate input representation that achieves near-BCJR performance and scales to jointly decode multiple noisy copies of the same codeword. We separately describe the input construction for the single copy scenario and the multiple copy scenario.

1) Single Copy: For $1 \le i \le n_{\text{in}}$, we define input tokens $\boldsymbol{Y}_i \in \mathbb{R}^{\delta \times q}$, over an alphabet of size q and a sliding window with size $\delta = d_{\max} - d_{\min} + 1$, where $d_{\min}$ and $d_{\max}$ serve as the lower and upper bounds on the drift of all received sequences. For each drift value j satisfying $d_{\min} \le j \le d_{\max}$ and for each alphabet symbol $\xi \in \mathbb{F}_q$, we define

$$Y_{i,j,\xi} = P(x_i^{\text{in}} = \xi) F(\xi, r_{i+d_j}). \tag{4}$$

Assuming that the outer codeword symbols are independent and identically distributed, the prior probability for any non-marker symbol x_i^{in} is $P(x_i^{\text{in}} = \xi) = \frac{1}{q}$. Consequently, the input tokens are $Y_{i,j,\xi} = \frac{1}{q}$ for all alphabet values ξ.

Let i_{m} denote the position of an inserted marker symbol $x_{i_{\text{m}}}^{\text{in}}$. Because the decoder knows the value ξ_{m} of this marker symbol, we get the prior probability $P(x_{i_{\text{m}}}^{\text{in}} = \xi_{\text{m}}) = 1$. Then the input tokens resolve to

$$Y_{i_{\text{m}},j,\xi} = \begin{cases} F(\xi_{\text{m}}, r_{i_{\text{m}}+d_j}) & \text{if } \xi = \xi_{\text{m}}, \\ 0 & \text{else.} \end{cases}$$

We flatten each token $\boldsymbol{Y}_i$ and then embed the flattened tokens into the transformer's hidden dimension d_{k} using a linear layer. Because the transformer is permutation-invariant, we add learned positional embeddings to each embedded token before passing them through the transformer. The transformer's output, with dimensions $n_{\text{in}} \times d_{\text{k}}$, is passed through a final linear layer of size $d_{\text{k}} \times 1$ and then through the sigmoid function $\sigma(x) = \frac{1}{1+\exp(-x)}$ to produce the prediction probabilities $\hat{\boldsymbol{x}}^{\text{in}} \in [0,1]^{n_{\text{in}}}$.

2) Multiple Sequence Alignment: The input representation extends naturally to handle multiple noisy copies of a transmitted codeword. Let M denote the number of transmissions. For the k-th transmission (with $1 \le k \le M$) and for symbol position $1 \le i \le n_{\text{in}}$, we define the token matrices $\boldsymbol{Y}_i^k \in \mathbb{R}^{\delta \times q}$ as in Equation (4) and apply the same embedding. We then concatenate the tensors $\boldsymbol{Y}^k$ along their first dimension and add a second positional embedding that encodes the sequence position k. In Appendix C, we demonstrate that this sequence position encoding reduces error rates.

The embedded input passes through the transformer and a final $d_{\text{k}} \times 1$ linear layer, as in the single-sequence case, yielding tokens $\hat{\boldsymbol{y}} \in \mathbb{R}^{M n_{\text{in}}}$. We compute $\hat{\boldsymbol{y}}^{\text{in}} \in \mathbb{R}^{n_{\text{in}}}$ as

$$\hat{y}_i^{\text{in}} = \frac{1}{M} \sum_{j=1}^{M} \hat{y}_{j n_{\text{in}} + i}, \tag{5}$$

since we found that simple averaging performs as well as a linear layer (see Appendix A). Finally, $\hat{\boldsymbol{y}}^{\text{in}}$ is passed through the sigmoid function σ to yield prediction probabilities $\hat{\boldsymbol{x}}^{\text{in}}$.

B. ECCT as Outer Decoder

We adapt ECCT for use as an outer decoder by transforming its input. We normalize the inner decoder's approximations of the a posterior probabilities $P(x_i^{\text{in}} = 0 \,|\, \boldsymbol{r})$ (as introduced in Equation (1)), which yields a probabilistic vector $\tilde{\boldsymbol{x}}^{\text{out}} \in [0,1]^n$. We then transform this vector into bipolar vectors defined by $\boldsymbol{x}^{\phi} = \phi(\tilde{\boldsymbol{x}}^{\text{out}}) \in [-1,1]^{n_{\text{out}}}$. Then the input to ECCT is the concatenation of the magnitude part $|\boldsymbol{x}^{\phi}|$ and the syndrome part $\phi\left(\text{syn}\left(\text{bin}\left(\boldsymbol{x}^{\phi}\right)\right)\right)$.

The inner decoder introduces noise that depends on the specific input codeword. As a result, outer decoding over the IDS channel is influenced by the codeword, unlike decoding over a binary-input symmetric-output channel where the noise is the same regardless of the input. For example, aligning an all-zero codeword is simpler than aligning a codeword with alternating patterns. Consequently, we cannot train only on the zero codeword as done in the article [7]; instead, we train on pseudorandomly generated codewords to capture the codeword-dependent noise effects.

C. ConvBCJRFormer

We propose *ConvBCJRFormer*, an extension of *BCJRFormer* that decodes transmissions of a zero-terminated binary convolutional $(n_{\text{c}}, 1, m)$ codeword over the IDS channel. We denote the input message length by n_{out} and define the codeword length as $n_{\text{in}} = n_{\text{c}}(n_{\text{out}} + m)$. After encoding, we add a random offset $\boldsymbol{o} \in \mathbb{F}_2^{n_{\text{in}}}$.

1) Input Construction: For inner marker codes, each symbol is encoded independently. In these cases, the symbol-wise prior information in the *BCJRFormer* is sufficient. In contrast, encoding a symbol x_i^{out} with a convolutional code depends not only on the symbol itself but also on the encoder's state,

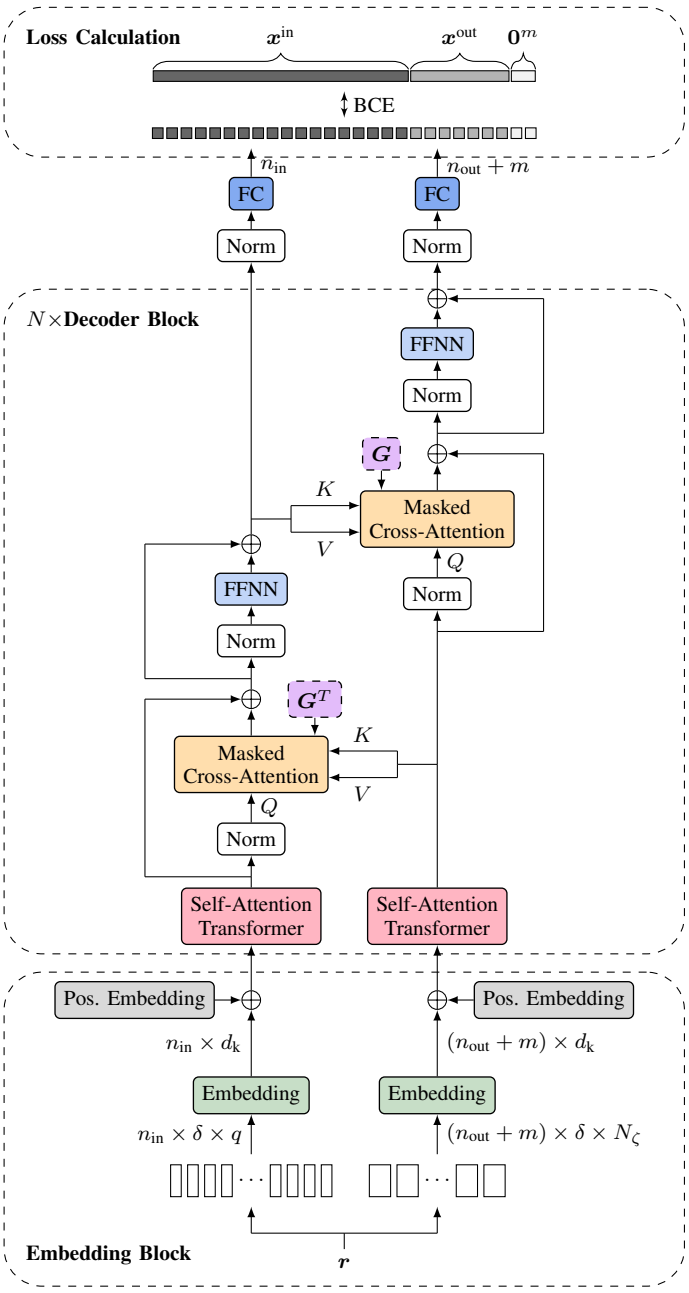


Fig. 5: Embedding, architecture, and loss calculation for *ConvBCJRFormer*. Visualization inspired by the article [24].

which is determined by the previous m symbols. Because the encoded symbols are generated based on the encoder's state, they are not independent. The symbol-wise input used in *BCJRFormer* does not capture the state dependency. Therefore, we extend the input in *ConvBCJRFormer* to include a state-wise representation.

We use two drift-based sliding windows: one for symbols, denoted by $\boldsymbol{Y}^{\text{symb}}$, and one for states, denoted by $\boldsymbol{Y}^{\text{state}}$. The symbol-based sliding window is identical to the one used for marker codes (see Equation (4)). The state-based sliding window is a tensor with dimensions $(n_{\text{out}}+m) \times \delta \times N_\zeta$. Here, $\delta = d_{\max} - d_{\min} + 1$ is the size of the sliding window, and $N_\zeta = 2^{n_{\text{c}}}$ is the number of possible output sequences $\boldsymbol{\zeta} \in \mathbb{F}_2^{n_{\text{c}}}$. These sequences correspond to a convolutional state transition from x_{i-1}^{out} to x_i^{out} in the input sequence, where $1 \le i \le (n_{\text{out}}+m)$. We denote the output sequence

$$[x^{\text{in}}_{n_{\text{c}}(i-1)+1}, x^{\text{in}}_{n_{\text{c}}(i-1)+2}, \dots, x^{\text{in}}_{n_{\text{c}}i}]$$

that arises from this state transition as $\boldsymbol{x}^{\text{out}}_{(i-1)\to i}$.

We enumerate all possible output sequences $\boldsymbol{\zeta}$ using indices $\zeta = 1, 2, \dots, N_\zeta$. Assume that the drift is d_j before the transmission of $\boldsymbol{x}^{\text{out}}_{(i-1)\to i}$. We define the scalar element $Y^{\text{state}}_{i,j,\zeta}$ as the joint probability that the transmitted state output $\boldsymbol{x}^{\text{out}}_{(i-1)\to i}$ equals $\boldsymbol{\zeta}$ and that the observed output is the sequence

$$[r_{n_{\text{c}}(i-1)+d_j+1}, r_{n_{\text{c}}(i-1)+d_j+2}, \dots, r_{n_{\text{c}}i+d_j}],$$

under the simplifying assumption that each transmitted symbol is independently substituted with probability p_S and that there are no insertions or deletions. When we include the additional offset vector $\boldsymbol{o}$ (with addition over $\mathbb{F}_2$), the joint probability becomes

$$\begin{aligned} Y^{\text{state}}_{i,j,\zeta} =& P([x^{\text{in}}_{n_{\text{c}}(i-1)+1}, x^{\text{in}}_{n_{\text{c}}(i-1)+2}, \dots, x^{\text{in}}_{n_{\text{c}}i}] = \boldsymbol{\zeta}) \\ & \cdot \prod_{l=1}^{n_{\text{c}}} F(x^{\text{in}}_{n_{\text{c}}(i-1)+l}, r_{n_{\text{c}}(i-1)+l+d_j} + o_{n_{\text{c}}(i-1)+l+d_j}), \end{aligned}$$

for $1 \le i \le (n_{\text{out}}+m)$, $d_{\min} \le j \le d_{\max}$, and $\zeta = 1, 2, \dots, N_\zeta$. For simplicity, we assume a balanced convolutional code. This means that, without knowing the encoder's state, all output sequences associated with a state transition $x_i^{\text{out}} \to x_{i+1}^{\text{out}}$ are equally likely. Then the prior probabilities become

$$P([x^{\text{in}}_{n_{\text{c}}(i-1)+1}, x^{\text{in}}_{n_{\text{c}}(i-1)+2}, \dots, x^{\text{in}}_{n_{\text{c}}i}] = \boldsymbol{\zeta}) = \frac{1}{N_\zeta}.$$

The state input representation $\boldsymbol{Y}^{\text{state}}$ links the received sequence to the output of convolutional state transitions. However, assuming that this output is only affected by substitutions is overly simplistic and fails to capture all the information in the received sequence. To overcome this limitation, we also include the symbol-level representation $\boldsymbol{Y}^{\text{symb}}$.

2) Architecture: For input codeword positions i, the encoded symbols $x^{\text{in}}_{n_{\text{c}}(i-1)+1}$, $x^{\text{in}}_{n_{\text{c}}(i-1)+2}$, $\dots$, $x^{\text{in}}_{n_{\text{c}}i}$ are all generated by the same input symbol x_i^{out} and the same convolutional state. The convolutional code's generator matrix G links these symbols to the state transition that generated them. We incorporate the generator matrix into *ConvBCJRFormer*'s attention mechanism. This integration connects the symbol representation with the state representation and helps the model learn the convolutional code's structure. Figure 5 illustrates the modified architecture.

We first embed the token representations $\boldsymbol{Y}^{\text{symb}}$ and $\boldsymbol{Y}^{\text{state}}$ separately. To synchronize these two input representations, we pass each representation through a separate vanilla transformer that uses unmasked self-attention blocks to capture global information.

To decode the synchronized representations, we use two masked cross-attention layers. We treat the synchronized

symbol representation as an abstract representation of the codeword and the synchronized state representation as an abstract representation of the input message, which we aim to recover.

First Cross-Attention Layer: The codeword representation attends to the message representation. Here, we derive the query matrix from the codeword representation and use the key and value matrices from the message representation. For $1 \leq i \leq n_{\text{in}}$ and $1 \leq j \leq (n_{\text{out}} + m)$, we mask the element $(\boldsymbol{Q}\boldsymbol{K}^T)_{ij}$ if the corresponding element in the transposed generator matrix $(\boldsymbol{G}^T)_{ij}$ is zero. If the element $(\boldsymbol{G}^T)_{ij}$ is non-zero, then the message symbol j was part of the convolutional state that produced codeword symbol i. Thus, the generator matrix incorporates information about the encoding process into the attention module.

Second Cross-Attention Layer: We repeat the process in the opposite direction. In this layer, the message representation acts as the query and the symbol representation serves as the key and value. This time, we use the untransposed generator matrix $\boldsymbol{G}$ as the mask. This design ensures that each token in the message representation attends only to those tokens in the codeword representation corresponding to symbols generated by the state that contains the message symbol.

Finally, we pass the outputs of the cross-attention layers for the symbol and state representation through distinct linear layers with dimensions $d_{\text{k}} \times 1$. We then apply a sigmoid function σ to produce the prediction probabilities $\hat{\boldsymbol{x}}^{\text{in}}$ for the inner codeword $\boldsymbol{x}^{\text{in}}$, $\hat{\boldsymbol{x}}^{\text{out}}$ for the outer codeword $\boldsymbol{x}^{\text{out}}$, and $\hat{\mathbf{0}}^m$ for the final memory state.

D. Training

We train all models on the binary alphabet using the binary cross-entropy loss:

$$BCE(\hat{\boldsymbol{x}}, \boldsymbol{x}) = -\frac{1}{n}\sum_{i=1}^{n} x_i \ln(\hat{x}_i) + (1 - x_i)\ln(1 - \hat{x}_i), \quad (6)$$

which measures the difference between the model predictions $\hat{\boldsymbol{x}}$ and the target sequence $\boldsymbol{x}$ (i.e., the inner or outer sequence to be decoded). Our training dataset is generated dynamically by generating pseudorandom sequences $\boldsymbol{m} \in \mathbb{F}_q^k$.

For *BCJRFormer*, each generated sequence is first encoded using an LDPC outer code and combined with an inner marker code. We then simulate M transmissions over the IDS channel. When using a binary alphabet, the model is optimized by minimizing the binary cross-entropy loss between the inner decoder's predictions $\hat{\boldsymbol{x}}^{\text{in}}$ and the corresponding inner codeword bits $\boldsymbol{x}^{\text{in}}$. For non-binary alphabets, we replace the binary cross-entropy loss with multi-class cross entropy.

The outer ECCT decoder is designed to learn the inner decoder's multiplicative noise. This noise is defined as

$$\boldsymbol{z} = \boldsymbol{x}^{\phi} \cdot \phi(\boldsymbol{x}^{\text{out}}),$$

and the decoder is trained by minimizing the loss $\text{BCE}(\hat{\boldsymbol{z}}, \text{bin}(\boldsymbol{z}))$, where $\hat{\boldsymbol{z}}$ is the ECCT's prediction. The final predicted outer codeword is then obtained by

$$\hat{\boldsymbol{x}}^{\text{out}} = \text{bin}(-\hat{\boldsymbol{z}} \cdot \text{sign}(\boldsymbol{x}^{\phi})).$$

For the convolutional decoder *ConvBCJRFormer*, we encode each generated sequence $\boldsymbol{m} \in \mathbb{F}_2^k$ using a convolutional code. A new pseudorandom offset $\boldsymbol{o}$ is generated for each transmission. We optimize the binary cross-entropy loss computed between the concatenated model predictions — $\hat{\boldsymbol{x}}^{\text{in}}$, $\hat{\boldsymbol{x}}^{\text{out}}$, and $\hat{\mathbf{0}}^m$ — and the concatenated target sequences — $\boldsymbol{x}^{\text{in}}$, $\boldsymbol{x}^{\text{out}}$ and $\mathbf{0}^m$.

E. Complexity

We analyze the complexity of jointly decoding multiple noisy copies of a codeword using the BCJR algorithm and *BCJRFormer* to justify our deep learning-based approach.

As in Section IV-A, we restrict the drift states of the BCJR decoder at each time step to the range from $d_{\min}$ to $d_{\max}$, where the total number of drift states is defined as $\delta = d_{\max} - d_{\min} + 1$. When decoding M copies jointly, the decoder must track δ^M distinct drift states. Each drift state has $\kappa = I_{\max} + 2$ possible transitions, where $I_{\max}$ is the maximum number of insertions allowed per symbol and the additional 2 accounts for a deletion or a transmission. As a result, the overall complexity of the BCJR decoding process is $O(n_{\text{in}}(\kappa\delta)^M)$, where n_{in} denotes the input sequence length [13].

In contrast, *BCJRFormer* scales only quadratically with the number of copies M. The conversion of the received sequences into the model's input (as detailed in Equation (4)) and the subsequent embedding have a combined complexity of $O(Mn_{\text{in}}\delta q d_{\text{k}})$. Each transformer block incurs quadratic complexity, specifically $O\left(M^2 n_{\text{in}}^2 N_{\text{dec}} d_{\text{k}}\right)$, where N_{dec} is the number of attention layers [29].

V. Experimental Results

We show that our proposed *BCJRFormer* model decodes multiple noisy copies of transmitted sequences with error rates comparable to joint BCJR decoding. Moreover, *BCJRFormer* scales to number of sequences that are computationally infeasible for the BCJR algorithm. We also demonstrate that combining *BCJRFormer* with ECCT yields better performance than using either the BCJR algorithm or Belief Propagation. Finally, we show that *ConvBCJRFormer* can jointly synchronize and decode convolutional codes with only minor trade-offs compared to the BCJR algorithm.

A. Setup

We train all neural decoders with the Adam optimizer [31]. Each *BCJRFormer* model has a hidden dimension of 96, six attention layers, and eight attention heads per layer, for a total of about 700,000 parameters. We train at a batch size of 256 for 160,000 iterations without dropout, and we use a cosine decay learning rate schedule that starts at $2.5\text{e}{-4}$ and decays to $2.5\text{e}{-5}$ after a warmup of 20,000 iterations.

All ECCT models have a hidden dimension of 128, with eight attention layers and eight attention heads per layer, for a total of about 1.6 million parameters. We train the models at a batch size of 1024 for 120,000 iterations using a constant learning rate of $1\text{e}{-4}$ and no dropout.

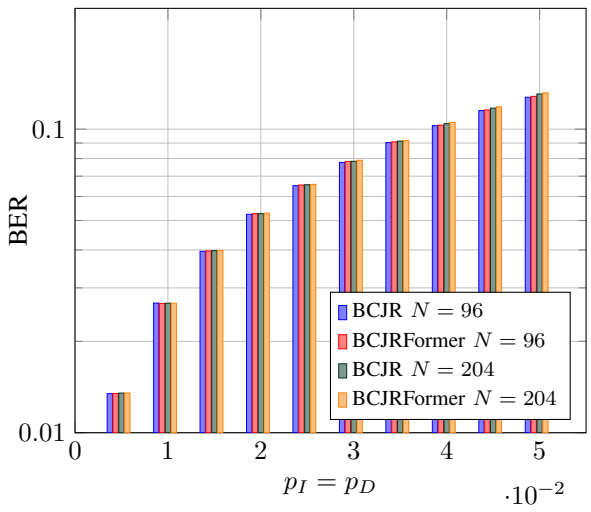


Fig. 6: *BCJRFormer* performs comparably to BCJR for single transmissions at $p_S = 0$. Outer LDPC codes of lengths $(96, 48)$ and $(204, 102)$ are concatenated with markers $\boldsymbol{s}_{\mathrm{m}} = 001$ inserted every $N_{\mathrm{m}} = 6$ bits.

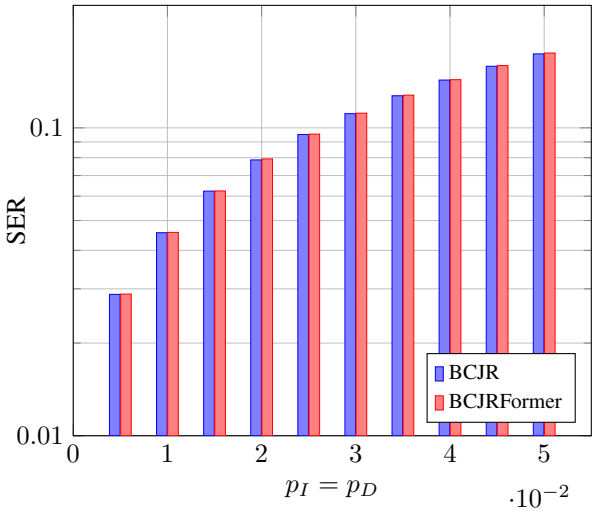


Fig. 7: For short quaternary codes, *BCJRFormer* achieves error rates comparable to BCJR. A protograph $(64, 32)$ LDPC code is transmitted with markers $\boldsymbol{s}_{\mathrm{m}} = 32$ inserted every $N_{\mathrm{m}} = 6$ symbols at $p_S = 0.012$.

For inner decoders we restrict the drift states $d_{\min}$ and $d_{\max}$ to $\pm 5\sqrt{n_{\mathrm{in}} \frac{p_D}{1-p_D}}$ and fix the maximum number of insertions per transmitted symbol to $I_{\max} = 2$ [3]. When we use Belief Propagation for outer decoding, we run 50 iterations to ensure convergence.

We measure performance using the bit error rate (BER) for binary transmissions and the symbol error rate (SER) for quaternary transmissions. The bit error rate and symbol error rate are defined as the ratio of incorrectly decoded bits or symbols to the total number of transmitted bits or symbols, respectively. We average error rates over 409,600 randomly generated codewords, unless noted otherwise.

B. BCJRFormer for Single Marker Codewords

We compare *BCJRFormer* and the BCJR algorithm for inner decoding of single binary sequences using a short $(96, 48)$ LDPC code [32] and a longer $(204, 102)$ LDPC code [33]. For the short code, we insert the marker sequence $\boldsymbol{s}_{\mathrm{m}} = 001$ every $N_{\mathrm{m}} = 6$ bits, and for the longer code, every $N_{\mathrm{m}} = 7$ bits. This yields inner codeword lengths of 144 and 291 with rates of $2/3$ and approximately 0.7, respectively. We trained several models over a range of deletion probabilities (set equal to the insertion probability) while fixing the substitution probability at $p_S = 0$. The bit error rate results in Figure 6 show that *BCJRFormer* closely matches the performance of the BCJR algorithm for both code lengths across all insertion and deletion rates.

C. BCJRFormer for Quaternary Marker Codewords

We also demonstrate that *BCJRFormer* performs well for non-binary codes. We use an outer $(64, 32)$ quaternary protograph LDPC code, as proposed in the paper [13]. The shortest cycle in the corresponding Tanner graph has a length of eight. The weights in the code's parity-check matrix were set uniformly at random once and then fixed for all experiments. For more details on the construction and the corresponding protograph, see Appendix G.

We insert the marker sequence $\boldsymbol{s}_{\mathrm{m}} = 32$ every $N_{\mathrm{m}} = 6$ symbols, resulting in an inner codeword length of $n_{\mathrm{in}} = 84$. We vary the insertion and deletion probabilities (with $p_I = p_D$) and fix the channel's substitution probability at $p_S = 0.012$. We compare the symbol error rate of the trained models with that of the BCJR algorithm. Figure 7 shows that *BCJRFormer* remains competitive across all channel configurations. For a more detailed analysis of the symbol-wise differences between the two decoders, see Appendix D.

D. BCJRFormer for Joint Inner Decoding

Next, we evaluate *BCJRFormer* for jointly decoding M noisy copies of a codeword. We train separate inner decoders for different values of M and for varying deletion and insertion probabilities $p_D = p_I$ with substitution probability $p_S = 0.012$, and we compare their error rates with those of the BCJR algorithm. For $M = 3$, we evaluate the BCJR algorithm's error rates over only 40,960 samples because its complexity grows exponentially. As shown in Figure 8, the error rates of *BCJRFormer* scale in a manner similar to those of the BCJR algorithm as the number of received copies increases. For values $M > 3$, *BCJRFormer* attains decreasing bit error rates, whereas the BCJR algorithm becomes computationally impractical.

In DNA data storage, the number of received sequences (denoted as M_k) varies for each transmitted sequence, which makes training models for fixed cluster sizes inefficient. We demonstrate that a single *BCJRFormer* model can decode sequences when the number of copies varies between $M_{\min}$ and $M_{\max}$.

To handle varying M_k, we pad the concatenated input — which originally has dimensions $n_{\mathrm{in}} M_k \times \delta \times q$ — with a dedicated token so that it has dimensions $n_{\mathrm{in}} M_{\max} \times \delta \times q$, and we mask the $M_{\max} - M_k$ padded columns in both the attention mechanism (3) and the mean aggregation (5).

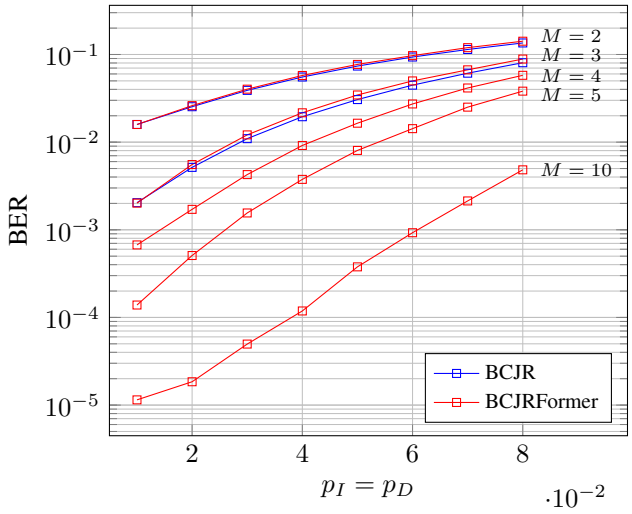


Fig. 8: Joint inner decoding performance of *BCJRFormer* scales similarly to that of the BCJR algorithm across various cluster sizes M. A $(96, 48)$ LDPC outer code is transmitted with markers $\boldsymbol{s}_{\mathrm{m}} = 001$ inserted every $N_{\mathrm{m}} = 6$ bits, at $p_S = 0.012$.

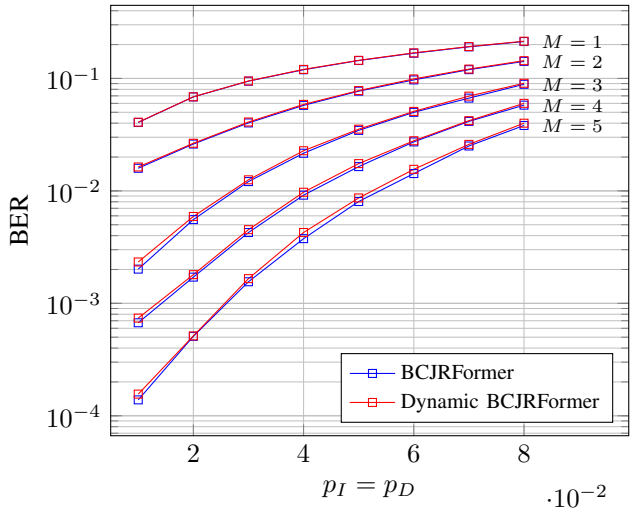


Fig. 9: *BCJRFormer* models trained with various cluster sizes perform comparably to those trained on fixed cluster sizes M. Markers $\boldsymbol{s}_{\mathrm{m}} = 001$ are inserted every $N_{\mathrm{m}} = 6$ bits into a $(96, 48)$ LDPC outer code, and transmitted at $p_S = 0.012$.

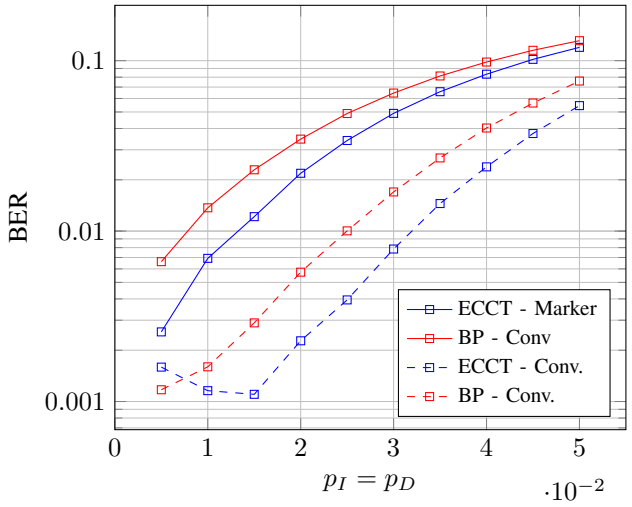


Fig. 10: The outer ECCT decoder outperforms Belief Propagation for a $(96, 48)$ LDPC outer code at $p_S = 0.012$, as shown for a rate-$1/2$ convolutional code ($g = [5, 7]_8$), and a 001 marker code with $N_{\mathrm{m}} = 6$.

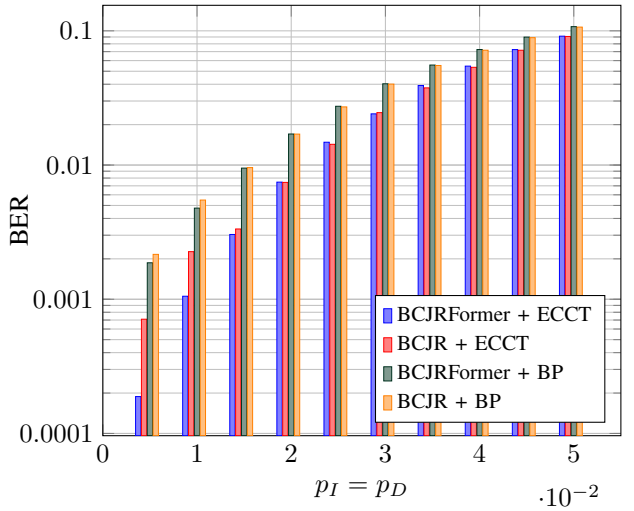


Fig. 11: Transformer-based decoder combinations achieve lower error rates than iterative algorithms, as shown at $p_S = 0.0$ with a $(96, 48)$ LDPC code and markers $\boldsymbol{s}_{\mathrm{m}} = 001$ inserted every $N_{\mathrm{m}} = 6$ bits.

We train dynamic models across a range of deletion and insertion probabilities ($p_I = p_D$) with substitution probability p_S by sampling M_k uniformly at random from the range $[M_{\min}, M_{\max}]$. We evaluate the dynamically trained model by comparing it to models trained on the same channel configuration with a fixed M_k. As shown in Figure 9, the dynamic model performs on par with models trained using a fixed M_k and even attains lower bit error rates in some scenarios.

E. ECCT for Outer Decoding

We compare the ECCT decoder with the Belief Propagation decoder using the short $(96, 48)$ LDPC code from Section V-B. We consider two inner code constructions. First, we concatenate the LDPC code with a $(2, 1, 2)$ zero-terminated convolutional code with generator polynomials $g = (5, 7)_8$. The codeword is combined with a pseudorandom offset which varies with each transmission. Secondly, we insert markers, $\boldsymbol{s}_{\mathrm{m}} = 001$, at a fixed interval N_{m}. We train models using a range of deletion and insertion probabilities (with $p_I = p_D$) and a fixed substitution probability $p_S = 0.012$ using the a posteriori probability approximations from an inner BCJR decoder. Figure 10 shows that the ECCT decoder outperforms Belief Propagation decoding across most channel configurations. We note that the performance gap between ECCT and Belief Propagation widens in the low probability domain. However, when using inner convolutional codes at extremely low deletion and insertion probabilities, the performance of ECCT deteriorates.

F. End-to-End Transformer Decoding

We demonstrate that an end-to-end transformer pipeline — using *BCJRFormer* as the inner decoder and ECCT as the outer decoder — outperforms pipelines that use either the BCJR algorithm or Belief Propagation as decoders. We reuse the *BCJRFormer* models from Section V-B and train each ECCT outer decoder on the outputs produced by the inner decoder for a channel configuration with equal insertion and deletion rates ($p_I = p_D$) and a substitution probability of $p_S = 0.012$. Figure 11 compares various combinations of inner and outer decoders. We observe that the ECCT yields lower error rates than Belief Propagation decoding across all probability ranges. In the high probability domain, the pipeline using the iterative BCJR inner decoder slightly outperforms the pipeline using *BCJRFormer*. In the low probability domain, all decoder combinations that use *BCJRFormer* as the inner decoder significantly outperform those that use the BCJR algorithm. These results shows that transformer-based pipelines can outperform iterative approaches in concatenated coding.

G. ConvBCJRFormer for Convolutional Decoding

We hypothesize that a major benefit of neural decoders is their ability to incorporate outer code information during synchronization which is computationally infeasible within the BCJR algorithm when applied to general linear codes [8]. In a first step toward incorporating linear code information, we consider the joint synchronization and decoding of linear convolutional codes.

We compare the error rates of our proposed *ConvBCJRFormer* decoder with those of the BCJR algorithm. We construct the input as described in Section IV-C, setting the drift window size δ of the state inputs equal to that of the symbol inputs. The *ConvBCJRFormer* models have a hidden dimension of $d_{\text{k}} = 96$ and consist of four decoder blocks. Both the symbol and state self-attention transformers have three attention layers, followed by a single cross-attention block. All attention layers have six heads. In total, each model has approximately 3.6 million learnable parameters.

We train each *ConvBCJRFormer* model for 480,000 iterations with a batch size of 512 and no dropout. We use a cosine decay learning rate schedule that starts at $2.75\text{e}{-4}$ and decays to $2.75\text{e}{-5}$ after a 20,000-iteration warmup. We encode a LDPC $(96, 48)$ outer codeword using a rate-$1/2$ zero-terminated $[5, 7]_8$ convolutional code, as introduced in Section IV-C. The resulting convolutional codeword has a length of $n_{\text{in}} = 196$. We fix the IDS channel's substitution rate at $p_S = 0.012$ and train the models over a range of equal insertion and deletion rates ($p_I = p_D$).

In Figure 12, we compare the difference in bit error rates between the decoded outer codeword $\hat{\boldsymbol{x}}^{\text{out}}$ and the outer codeword $\boldsymbol{x}^{\text{out}}$ for both the BCJR algorithm and the *ConvBCJRFormer* decoder. While error rates of our proposed model are slightly higher than those of the BCJR algorithm, we observe that our proposed architecture successfully learns the convolutional code structure. This becomes more clear when comparing our model with models that do not employ any attention masking, as we demonstrate in Appendix F.

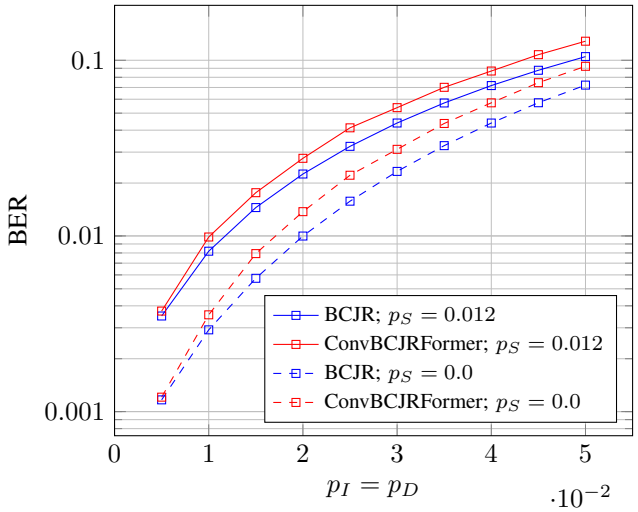


Fig. 12: Error rates of *ConvBCJRFormer* are marginally higher than those of the BCJR algorithm, shown for a rate-$1/2$ convolutional code with $g = [5, 7]_8$ and an input codeword length of $n_{\text{out}} = 96$.

VI. Conclusion

In this work, we propose a high-performance and efficient two-step transformer-based decoding approach for handling multiple noisy copies of a codeword transmitted over the IDS channel.

By replacing the BCJR algorithm with *BCJRFormer* as the inner decoder and substituting Belief Propagation with ECCT as the outer decoder, our end-to-end pipeline achieves lower error rates compared to iterative algorithms while efficiently scaling to cluster sizes that are not feasible with the BCJR algorithm. Our study is currently limited to randomly generated sequences. A promising direction for future study is to apply and fine-tune our methodology on real DNA traces. In this context, it would be valuable to compare the performance of *BCJRFormer* for coded multiple sequence alignment with other methods, such as those proposed in the papers [13], [14]. We further introduced *ConvBCJRFormer*, a transformer-based architecture that jointly synchronizes and decodes convolutional codes. A natural extension for future work is to incorporate the structure of more general linear codes into the inner decoding process. This approach could improve synchronization and overcome the computational limitations of the BCJR algorithm.

REFERENCES

[1] R. Heckel, G. Mikutis, and R. N. Grass, "A Characterization of the DNA Data Storage Channel," *Scientific Reports*, vol. 9, no. 1, p. 9663, Jul. 2019.

[2] F. Sellers, "Bit loss and gain correction code," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 35–38, Jan. 1962.

[3] M. Davey and D. Mackay, "Reliable communication over channels with insertions, deletions, and substitutions," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 687–698, Feb. 2001.

[4] F. Wang, D. Fertonani, and T. M. Duman, "Symbol-Level Synchronization and LDPC Code Design for Insertion/Deletion Channels," *IEEE Transactions on Communications*, vol. 59, no. 5, pp. 1287–1297, May 2011.

[5] I. Tal, H. D. Pfister, A. Fazeli, and A. Vardy, "Polar Codes for the Deletion Channel: Weak and Strong Polarization," *IEEE Transactions on Information Theory*, vol. 68, no. 4, pp. 2239–2265, Apr. 2022.

[6] R. Tanner, "A recursive approach to low complexity codes," *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, 1981.

[7] Y. Choukroun and L. Wolf, "Error correction code transformer," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 38 695–38 705.

[8] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (Corresp.)," *IEEE Transactions on Information Theory*, vol. 20, no. 2, pp. 284–287, 1974.

[9] L. Organick, S. D. Ang, Y.-J. Chen, R. Lopez, S. Yekhanin, K. Makarychev, M. Z. Racz, G. Kamath, P. Gopalan, B. Nguyen, C. N. Takahashi, S. Newman, H.-Y. Parker, C. Rashtchian, K. Stewart, G. Gupta, R. Carlson, J. Mulligan, D. Carmean, G. Seelig, L. Ceze, and K. Strauss, "Random access in large-scale DNA data storage," *Nature Biotechnology*, vol. 36, no. 3, pp. 242–248, Mar. 2018.

[10] D. Bar-Lev, I. Orr, O. Sabary, T. Etzion, and E. Yaakobi, "Deep DNA storage: Scalable and robust DNA storage via coding theory and deep learning," 2024.

[11] M. Girsch and R. Heckel, "Trace reconstruction for DNA data storage using language models," 2025. [Online]. Available: https://openreview.net/forum?id=rkfiJQMFcw

[12] P. L. Antkowiak, J. Lietard, M. Z. Darestani, M. M. Somoza, W. J. Stark, R. Heckel, and R. N. Grass, "Low cost DNA data storage using photolithographic synthesis and advanced information reconstruction and error correction," *Nature Communications*, vol. 11, no. 1, p. 5345, Oct. 2020.

[13] I. Maarouf, A. Lenz, L. Welter, A. Wachter-Zeh, E. Rosnes, and A. G. i Amat, "Concatenated codes for multiple reads of a DNA sequence," *IEEE Transactions on Information Theory*, vol. 69, no. 2, pp. 910–927, 2023.

[14] S. R. Srinivasavaradhan, S. Gopi, H. D. Pfister, and S. Yekhanin, "Trellis BMA: Coded trace reconstruction on IDS channels for DNA storage," in *2021 IEEE International Symposium on Information Theory (ISIT)*, 2021, pp. 2453–2458.

[15] E. Nachmani, Y. Be'ery, and D. Burshtein, "Learning to decode linear codes using deep learning," in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Sep. 2016, pp. 341–346.

[16] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Beery, "Deep Learning Methods for Improved Decoding of Linear Codes," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 119–131, Feb. 2018.

[17] M. H. Sazlı and C. Işık, "Neural network implementation of the BCJR algorithm," *Digital Signal Processing*, vol. 17, no. 1, pp. 353–359, 2007.

[18] N. Shlezinger, N. Farsad, Y. C. Eldar, and A. J. Goldsmith, "Data-driven factor graphs for deep symbol detection," in *2020 IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 2682–2687.

[19] N. Farsad, N. Shlezinger, A. J. Goldsmith, and Y. C. Eldar, "Data-driven symbol detection via model-based machine learning," in *2021 IEEE Statistical Signal Processing Workshop (SSP)*, 2021, pp. 571–575.

[20] N. Shlezinger, Y. C. Eldar, N. Farsad, and A. J. Goldsmith, "ViterbiNet: Symbol detection using a deep learning based viterbi algorithm," in *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2019, pp. 1–5.

[21] N. Shlezinger, J. Whang, Y. C. Eldar, and A. G. Dimakis, "Model-based deep learning," *Proceedings of the IEEE*, vol. 111, no. 5, pp. 465–499, 2023.

[22] G. Ma, X. Jiao, J. Mu, H. Han, and Y. Yang, "Deep learning-based detection for marker codes over insertion and deletion channels," *IEEE Transactions on Communications*, vol. 72, no. 10, pp. 5945–5959, 2024.

[23] E. U. Kargı and T. M. Duman, "A deep learning based decoder for concatenated coding over deletion channels," in *ICC 2024 - IEEE International Conference on Communications*, 2024, pp. 2797–2802.

[24] S.-J. Park, H.-Y. Kwak, S.-H. Kim, Y. Kim, and J.-S. No, "CrossMPT: Cross-attention message-passing transformer for error correcting codes," in *The Thirteenth International Conference on Learning Representations*, 2025.

[25] R. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[26] V. Buttigieg and N. Farrugia, "Improved bit error rate performance of convolutional codes with synchronization errors," in *2015 IEEE International Conference on Communications (ICC)*, 2015, pp. 4077–4082.

[27] L. Bahl and F. Jelinek, "Decoding for channels with insertions, deletions, and substitutions with applications to speech recognition," *IEEE Transactions on Information Theory*, vol. 21, no. 4, pp. 404–411, 1975.

[28] F. Jelinek, *Statistical Methods for Speech Recognition*. Cambridge, MA, USA: MIT Press, 1998.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[30] A. Bennatan, Y. Choukroun, and P. Kisilev, "Deep learning for decoding of linear codes - a syndrome-based approach," in *2018 IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 1595–1599.

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd international conference on learning representations, ICLR 2015, san diego, CA, USA, may 7-9, 2015, conference track proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

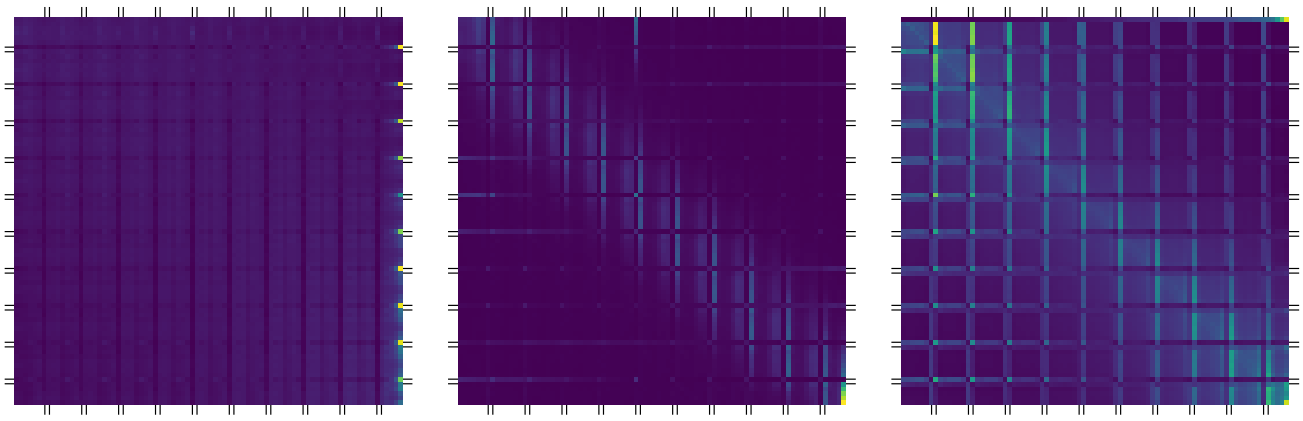
[32] M. Helmling, S. Scholl, F. Gensheimer, T. Dietz, K. Kraft, S. Ruzika, and N. Wehn, "Database of Channel Codes and ML Simulation Results," www.uni-kl.de/channel-codes, 2019.

[33] D. Mackay, "Encyclopedia of Sparse Graph Codes," https://www.inference.org.uk/mackay/codes/data.html.

[34] N. J. A. Sloane, "On single-deletion-correcting codes," in *Codes and Designs*, K. Arasu and Á. Seress, Eds. Berlin, New York: De Gruyter, 2002, pp. 273–292.

[35] M. Mitzenmacher, "A survey of results for deletion channels and related synchronization channels," *Probability Surveys*, vol. 6, no. none, pp. 1–33, 2009.

[36] B. K. Butler and P. H. Siegel, "Bounds on the minimum distance of punctured quasi-cyclic LDPC codes," *IEEE Transactions on Information Theory*, vol. 59, no. 7, pp. 4584–4597, 2013.



(a) Layer 1 (b) Layer 4 (c) Layer 6

Fig. 13: *BCJRFormer*'s attention on marker positions increases with layer depth. Shown are the average attention head scores (after applying the Softmax function in Equation (3)), where brighter colors indicate higher attention. Tick marks align with the positions of markers $\boldsymbol{s}_{\text{m}} = 23$ inserted into a quaternary $(64, 32)$ LDPC outer code. Results are averaged over 409,600 transmissions through an IDS channel with $p_I = p_D = 0.01$ and $p_S = 0.012$.

Appendix A
Attention Visualization

Figure 13 shows attention heatmaps at different depths of the *BCJRFormer* model, which was trained on the quaternary code described in Section V-C. We observe that in higher layers, attention is distributed globally. However, the marker positions of Layer 1 (Subfigure 13a) show very strong attention to the final sequence token. In the BCJR algorithm, received sequence length n_{rec} relative to the codeword length n_{in} is used to initialize the backward recursion. We conjecture that the model similarly uses the final token to extract early information about the severity of synchronization loss throughout the sequence.

We further note that the final symbol of the sequence is considerably easier to decode than other (non-marker) symbols (see, for example, Figure 16), which may further encourage attention. In deeper layers (Subfigures 13b and 13c), we observe very local attention that is directed toward nearby marker tokens. This is expected, since the markers provide the main source of prior information that can be used to synchronize the sequence.

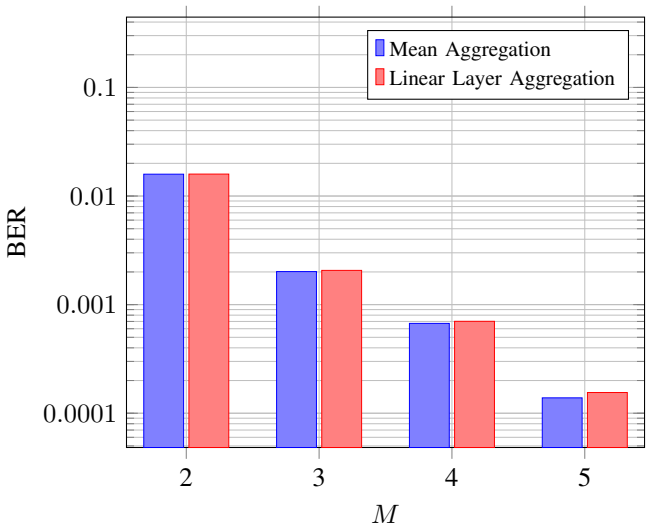


Fig. 14: Mean aggregation achieves lower error rates than a single linear layer for output dimensionality reduction in joint decoding with *BCJRFormer*. Models are trained on transmissions with $p_I = p_D = 0.01$ and $p_S = 0.012$ using an outer $(96, 48)$ LDPC code with inner markers $\boldsymbol{s}_{\text{m}} = 001$ inserted every $N_m = 6$ bits.

Appendix B
Ablation: Multiple Sequence Aggregation

When more than one sequence is transmitted (i.e., $M > 1$), the transformer block outputs a tensor of dimension Mn_{in} instead of n_{in}. To reduce the dimension to the inner code size n_{in}, we propose mean aggregation (see Equation (5)). An alternative

is to use a single linear layer. Because a linear layer introduces learnable parameters, it may lead to slight performance improvements. We compare the bit error rate of mean aggregation and linear layer aggregation for $M \in \{2, 3, 4, 5\}$. We use the shorter code construction described in Section V-B and set $p_I = p_D = 0.01$ and $p_S = 0.012$. Figure 14 shows that mean aggregation yields lower error rates for all values of M. The performance difference becomes more pronounced as the number of sequences increases. We believe that mean aggregation naturally preserves the significance of each input symbol, whereas a linear layer must learn that every token is meaningful.

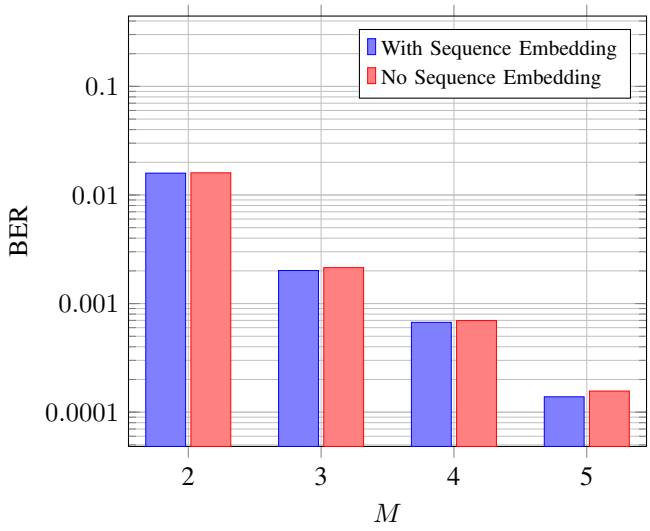


Fig. 15: Adding a sequence embedding improves error rates for jointly decoding multiple noisy copies of a codeword transmitted through a channel with $p_I = p_D = 0.01$ and $p_S = 0.012$. Shown here for an outer $(96, 48)$ LDPC code with markers $\boldsymbol{s}_{\mathrm{m}} = 001$ inserted every $N_m = 6$ bits.

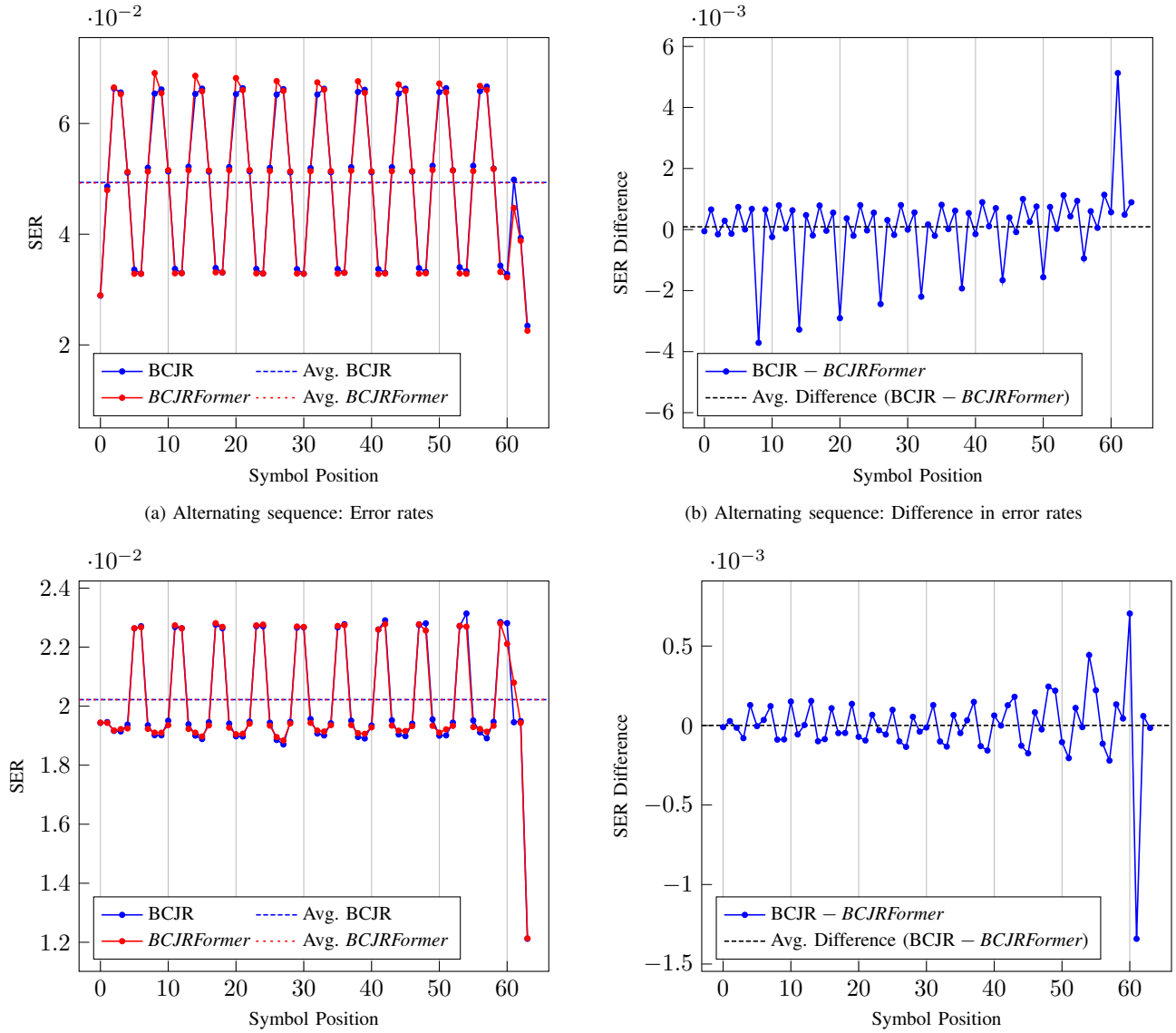
APPENDIX C
ABLATION: SEQUENCE EMBEDDING

As introduced in Section IV-A, we add a sequence embedding in addition to a positional embedding that indicates the sequence to which each symbol belongs. In Figure 15, we compare two models: one without sequence embedding and one with sequence embedding. We use the same code construction as in Section V-B, namely a $(96, 48)$ LDPC code concatenated with markers $\boldsymbol{s}_{\mathrm{m}} = 001$ inserted every $N_{\mathrm{m}} = 6$ bits. We fix the channel configuration to $p_I = p_D = 0.01$ and $p_S = 0.012$ and compare models across $M \in \{2, 3, 4, 5\}$ transmissions. Sequence embedding decreases error rates even for $M = 2$, and the figure suggests that the improvement becomes more pronounced as the number of sequences M increases.

APPENDIX D
OUTPUT DISTRIBUTION: COMPARISON OF BCJR AND BCJRFORMER

For transmissions over a deletion channel, the size of the deletion error balls (i.e., the number of sequences that can arise by deleting a fixed number of symbols) increases with the number of runs in the transmitted sequence. A run is defined as a block of consecutive symbols in a sequence (for example, the sequence 002111 has three runs). A maximum likelihood decoder can distinguish between two received sequences if their corresponding error balls do not intersect. Intuitively, transmitting an alternating codeword (e.g. $010101\ldots$) is more difficult to synchronize than transmitting the all-zero codeword because the alternating sequence has n_{in} runs, while the all-zero sequence has only one run [34], [35].

We explore the symbol-wise error rate distribution of *BCJRFormer* by comparing it to the BCJR algorithm for transmissions of these two codewords in the quaternary domain. We concatenate each sequence of length 64 with markers $\boldsymbol{s}_{\mathrm{m}} = 32$ inserted every $N_{\mathrm{m}} = 6$ symbols, and transmit them via an IDS channel with $p_I = p_D = 0.01$ and $p_S = 0.012$. We reuse the corresponding model from Section V-C and consider the same 40,960,000 transmissions of each sequence for both decoders. Figure 16 shows (left column) the symbol error rates for *BCJRFormer* and the BCJR algorithm, as well as the difference between the error rate of BCJR and that of *BCJRFormer* (right column). As expected, the average error rates of the alternating sequence (first row) are much higher than those of the all-zero sequence (second row). The error rate of symbols is strongly correlated with the distance from the nearest inserted marker.



(a) Alternating sequence: Error rates

(b) Alternating sequence: Difference in error rates

(c) All-zero sequence: Error rates

(d) All-zero sequence: Difference in error rates

Fig. 16: Error distributions for the alternating and all-zero sequences — shown in Subfigures (a) and (c), respectively — differ at specific positions between *BCJRFormer* and BCJR at specific positions, despite similar overall error rates. Subfigures (b) and (d) show the error rate differences, where values > 0 indicate that *BCJRFormer* outperforms BCJR and values < 0 indicate the reverse. The alternating and all-zero sequence of length 64, concatenated with markers $\boldsymbol{s}_{\mathrm{m}} = 32$ inserted every $N_{\mathrm{m}} = 6$ symbols, were transmitted through a channel with $p_S = 0.012$ and $p_D = p_I = 0.01$.

Appendix E
Ablation: Failure to Decode in Low Probability Domain for Vectorized Inputs

In Section IV-A1, we introduced the tensor input for *BCJRFormer*. More specifically, for $1 \leq i \leq n_{\mathrm{in}}$, we presented the construction of matrices $\boldsymbol{Y}_i \in \mathbb{R}^{\delta \times q}$ in Equation (4). For binary marker codes, we consider an alternative construction in which the input is summed. In this construction, we define $\boldsymbol{Y}_i^v \in \mathbb{R}^{\delta}$ as

$$Y_{i,j}^v = \sum_{\xi \in \{0,1\}} P(x_i^{\mathrm{in}} = \xi) F(\xi, r_{i+d_j}). \tag{7}$$

For non-marker positions i, we have $Y_{i,j}^v = 0.5$. For marker symbols with the value ξ_m, we have $Y_{i,j}^v = F(\xi_m, r_{i+d_j})$. If every non-marker symbol is covered by the window associated with a marker symbol (i.e., $\delta \geq N_{\mathrm{m}}$), it is unclear whether the aggregated input representation performs worse than the proposed one. For $p_I = p_D \geq 0.015$, the aggregated input performs equally well to the proposed one. Figure 17 shows the error rates for $p_I = p_D \leq 0.015$ at $p_S = 0.012$ for models trained

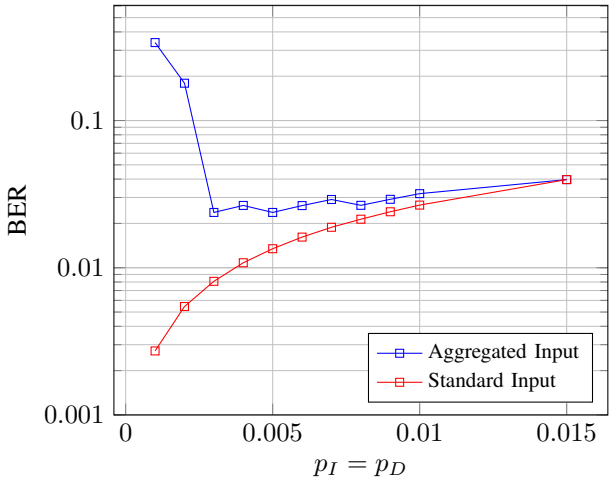


Fig. 17: Models using aggregated inputs fail to synchronize sequences when error probabilities are very low. The figure shows results for transmissions at $p_S = 0.012$ using a $(96, 48)$ LDPC outer code with markers $\boldsymbol{s}_{\mathrm{m}} = 001$ inserted at fixed intervals $N_{\mathrm{m}} = 6$.

with the proposed input and with the aggregated input. We reused the model and shorter concatenated code construction from Section V-B.

Models trained on the aggregated input generally perform worse and do not improve for deletion and insertion probabilities below 0.01. Furthermore, their performance deteriorates considerably for probabilities $p_D = p_I < 0.003$. We conjecture that the vectorized input leads to a much sparser representation of synchronization errors. When combined with a very low probability of synchronization errors, this results in the model having insufficient error representation to learn how to synchronize general sequences effectively. Our proposed input does not have this issue, because it also captures synchronization errors within tokens that are not at marker positions.

Appendix F
Training Convergence of ConvBCJRFormer With and Without Masking

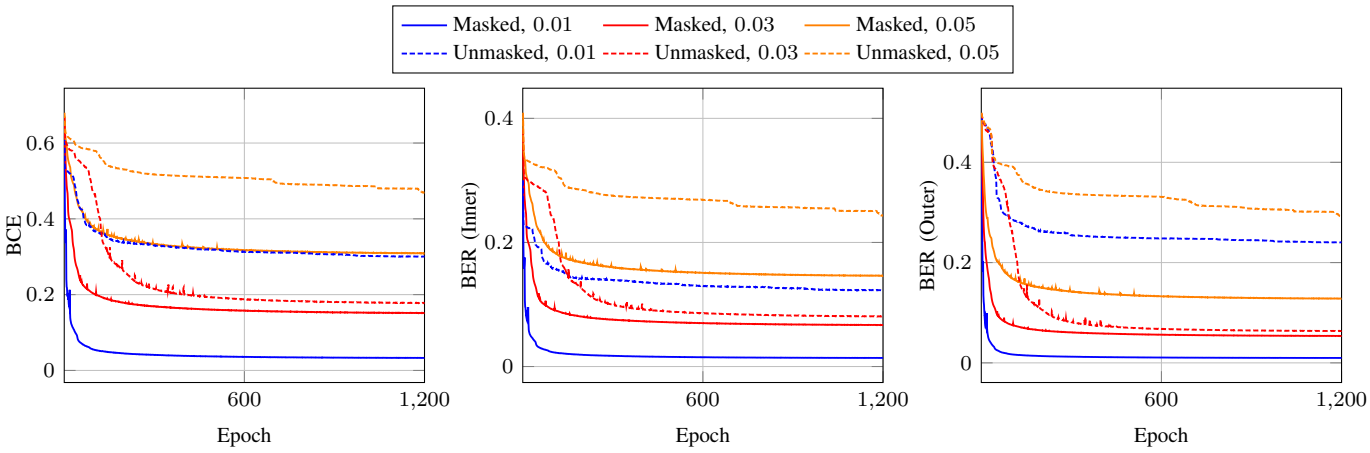


Fig. 18: Unmasked models fail to capture the convolutional structure. We compare metrics for insertion/deletion probabilities $p_I = p_D \in \{0.01, 0.03, 0.05\}$ at a fixed substitution probability of $p_S = 0.012$. The left graph shows the overall binary cross-entropy error, the middle graph the bit error rate between the inner codeword $\boldsymbol{x}^{\mathrm{in}}$ and its prediction $\hat{\boldsymbol{x}}^{\mathrm{in}}$, and the right graph the bit error rate between the outer codeword $\boldsymbol{x}^{\mathrm{out}}$ and its prediction $\hat{\boldsymbol{x}}^{\mathrm{out}}$. One epoch corresponds to 400 iterations.

We evaluate the impact of our proposed masking of *ConvBCJRFormer*'s cross-attention mechanism using the generator matrix G. In Figure 18, we compare the training convergence of models from Experiment V-G with equivalent models trained without masking. We observe that using the generator matrix G to exclude unrelated symbols and states from the attention mechanism yields much better final error rates. Furthermore, we observe that for certain probability ranges — such as when $p_I = p_D = 0.01$ — the training of unmasked models converges to a suboptimal state, presumably a local minimum. This premature convergence results in higher error rates than those observed for models trained on sequences with higher deletion/insertion error probabilities.

APPENDIX G
LDPC QUATERNARY PROTOGRAPH CONSTRUCTION

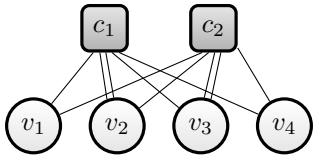


Fig. 19: Protograph $\boldsymbol{P}$ used to construct the parity-check matrix $\boldsymbol{H}$ in Section V-C

For reproducibility, we present the protograph used in the experiments in Section V-C. A protograph is a small Tanner graph, given by a matrix $\boldsymbol{P}$ with m_p rows corresponding to check nodes and n_p columns corresponding to variable nodes. Each entry $p_{i,j}$ gives the number of edges between check node i and variable node j. For our experiment, we use the protograph proposed in the paper [13], given by

$$\boldsymbol{P} = \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \end{bmatrix}.$$

Figure 19 shows the corresponding graph. A full parity-check matrix is derived by lifting the protograph. First, the protograph is replicated a specified number of times; then, the edges are permuted so that both the degree and interconnectivity between check nodes and variable nodes are preserved [36]. For non-binary codes, the weights of the edges in the resulting parity-check matrix can be chosen randomly. In our experiments, we randomly initialized the weights once and then fixed them for all experiments.