Efficient Transformer-based Decoder for Varshamov-Tenengolts Codes

Yali Wei,[*] Alan J.X. Guo,[†] and Yufan Dai[‡]

Center for Applied Mathematics, Tianjin University,Tianjin, China

Zihui Yan[§]

Frontiers Science Center for Synthetic Biology and Key Laboratory of Systems Bioengineering (Ministry of Education),
School of Chemical Engineering and Technology, Tianjin University, China

(Dated: March 3, 2025)

In recent years, the rise of DNA data storage technology has brought significant attention to the challenge of correcting insertion, deletion, and substitution (IDS) errors. Among various coding methods for IDS correction, Varshamov-Tenengolts (VT) codes, primarily designed for single-error correction, have emerged as a central research focus. While existing decoding methods achieve high accuracy in correcting a single error, they often fail to correct multiple IDS errors. In this work, we observe that VT codes retain some capability for addressing multiple errors by introducing a transformer-based VT decoder (TVTD) along with symbol- and statistic-based codeword embedding. Experimental results demonstrate that the proposed TVTD achieves perfect correction of a single error. Furthermore, when decoding multiple errors across various codeword lengths, the bit error rate and frame error rate are significantly improved compared to existing hard decision and soft-in soft-out algorithms. Additionally, through model architecture optimization, the proposed method reduces time consumption by an order of magnitude compared to other soft decoders.

INTRODUCTION.

DNA has recently attracted widespread attention as a long-term and high-density data storage medium [1–4]. An important challenge in DNA information encoding is the correction of various errors generated during DNA synthesis and sequencing, particularly insertion, deletion, and substitution (IDS) errors [5–7]. These IDS errors not only affect data integrity but also potentially compromise the reliability of storage and retrieval processes. Therefore, effectively addressing this issue has become a significant research topic in the field of DNA data storage.

To correct IDS errors, several coding methods have been adopted, including convolutional codes [8–10], watermark codes [11], time-varying block codes [12], and Varshamov-Tenengolts (VT) codes [13]. Among these methods, VT codes are considered an effective error correction scheme for DNA storage due to their asymptotic optimality. Specifically, VT codes require only $\lceil \log n \rceil + 1$ redundant bits to correct a single IDS error, making them highly efficient in theory. When a single IDS error occurs in VT codewords, traditional hard-decision (HD) decoding methods can effectively restore the original information [13–15]. However, in practical applications, multiple random IDS errors may be introduced, causing a significant deterioration in the performance of hard-decision decoding, thus severely affecting decoding accuracy and efficiency.

To address this problem, Yan et al. proposed a soft-in soft-out (SISO) decoding algorithm [16], which improves decoding performance by incorporating probabilistic in-formation. Compared to hard-decision decoding, SISO methods offer better robustness and error correction capability when handling multiple errors. However, despite the improvements of SISO methods, they still face bottlenecks in computational accuracy and speed when processing a large number of errors, particularly when dealing with longer codewords.

To overcome these limitations, this paper proposes a transformer-based VT decoder (TVTD) . The transformer architecture ([17]) can effectively capture complex relationships between input data, enabling it to better handle multiple IDS errors. Experimental results show that, compared to traditional decoding methods, TVTD can effectively decode multiple errors and accelerate the decoding processes through parallel computing, thereby significantly improving decoding efficiency. Specifically, the main contributions of this paper are as follows:

- The first deep learning-based binary VT code decoder: We use symbol- and statistic-based codeword embedding as positional encoding, and leverage the self-attention and cross-attention mechanisms in the transformer decoder to study the relationships between bits in VT codewords for efficient decoding.

- Significant improvement in decoding performance: Compared to existing methods, TVTD achieved the best experimental results across various codeword lengths, especially with long codewords. Specifically, as the codeword length varies, the bit error rate (BER) decreased by 2% to 20%, and the frame error rate (FER) decreased by 20% to 86%.

- Faster computational speed: By optimizing the model, TVTD accelerates training speed by 40% compared to the full transformer models. Additionally, in practical tests, the decoding speed of TVTD increased by over 5 times compared to SISO decoding, and for long codewords, the decoding speed improved by over 46 times, greatly enhancing its potential for large-scale data storage applications.

RELATED WORK.

The decoding of VT codewords with IDS errors has long been a central research topic in the field of information coding, particularly in terms of decoding accuracy and efficiency. To address this issue, N. J. A. Sloane et al. proposed an effective decoding method based on the weights and checksums of the received sequence. This method accurately locates errors by evaluating the weights of the received sequence and combining the checksums for correction, enabling the effective detection and correction of single IDS errors [13]. However, when multiple errors occur in VT codewords, this HD decoding method experiences a significant decline in accuracy.

To address this problem, Yan et al. proposed an improved algorithm based on the SISO decoding strategy [16]. This method relies on the Bahl-Cocke-Jelinek-Raviv (BCJR) algorithm and adopts the bitwise maximum a posteriori (MAP) decoding strategy. Compared to traditional hard decision decoding, the SISO method firstly tackled multiple errors correction. However, when the number of errors increases, the decoding accuracy declines. Moreover, the high computational complexity makes it difficult to be applied in real practice.

Error Correcting Codes (ECC) improve data reliability by adding redundant bits, enabling the detection and correction of errors during transmission or storage. Common ECCs include Hamming code, BCH code, LDPC code, and Turbo code, which are widely used to ensure data integrity [18]. In recent years, the rapid development of deep learning technology has brought revolutionary breakthroughs to the field of deep learning-based ECC, where Belief Propagation (BP) decoders have gradually become the main solution [19]. Compared to traditional decoding methods, such BP decoders based on neural networks not only improve decoding efficiency but also significantly enhance the systems robustness to noise interference [20, 21]. Considering the outstanding performance of transformer models in various tasks [17], they have also been gradually introduced into the research of ECC. Choukroun et al. advanced this field by utilizing transformer models for decoding linear codewords [22–24]. Their research demonstrated that transformer models not only improve decoding accuracy but also reduce the computational complexity of decoding, signifi-

cantly surpassing existing state-of-the-art decoding technologies.

However, despite the success of transformer models in linear codes, their application in VT code decoding remains a relatively unexplored research area.

BACKGROUND

In this study, we focus on decoding binary VT codewords. The following provides background on VT codes and related decoding algorithms that correct IDS errors.

VT code

This study focuses on the original structure of VT codes, which are a class of algebraic block codes composed of all binary vectors of length n that satisfy the following condition:

$$\text{VT}_{a,m}(n) = \left\{ v \in \{0,1\}^n : \sum_{i=1}^{n} i v_i \equiv a \pmod{m} \right\}. \quad (1)$$

Here, m is a predefined integer, and a is an integer satisfying $0 \leq a \leq m-1$, typically referred to as the parity of the sequence v. When $m \geq n+1$, the code can correct a deletion or insertion error, and when $m \geq 2n$, it can correct an IDS error [13]. In this work, the VT codes that correct IDS errors ($m = 2n + 1$) are investigated.

VT encoder

A systematic encoding scheme for VT codes [25], referred to as the VT encoder, is summarized as follows: For any message sequence $u = \{u_1, u_2, \ldots, u_p\} \in \{0,1\}^p$, the VT encoder inserts these information bits into a codeword $v = VT(u) \in VT_{a,2n+1}(n)$, where $y = n - \lceil \log n \rceil - 1$. The encoder inserts parity check bits at binary positions v_{2^i}, for $0 \leq i \leq n - y - 2$, and v_n. The parities are generated with the message symbols to ensure:

$$\sum_{i=1}^{n} i v_i \equiv a \pmod{2n+1}. \quad (2)$$

Example 1. *Given a message sequence $u = 11011$ with a fixed parity check $a = 0$, we have $n = 10, y = 5$, and $m = 21$. The codeword $v = (v_1, v_2, \ldots, v_{10})$ must satisfy: $\sum_{i=1}^{10} i v_i \equiv \sum_{i=1}^{4} 2^{i-1} v_{2i-1} + 1 \cdot 3 + 1 \cdot 5 + 0 \cdot 6 + 1 \cdot 7 + 1 \cdot 9 + 10 \cdot v_{10} \equiv 0 \pmod{21}$, we have $\sum_{i=1}^{4} 2^{i-1} v_{2i-1} + 10 \cdot v_{10} = 18$. This means $v_{10} = 1$ and $v_8 = 1$, and $v_1 = v_2 = v_4 = 0$. By expanding $18 - 10 = 8$ into binary form $1 \cdot 2^3$, the final codeword is $\bar{0}0\bar{1}\bar{0}101\bar{1}1\bar{1}$, where the overlined positions are parity check bits.*

Hard decision decoder

The HD decoder is used to detect and correct a single error in the codeword. The basic steps are as follows:

- For a received sequence is $r = (r_1, r_2, \ldots, r_N)$, the parity check condition $\bar{a} = \sum_{i=1}^{N} i \cdot r_i \pmod{2n+1}$ is checked. If this condition is not satisfied, the codeword is corrupted.

- The received sequence r is a corrupted form of the original sequence, represented as $r = [v_1, \ldots, v_{t-1}, p, v_t, \ldots, v_n]$, where $v_1, \ldots, v_{t-1}$ is the part before the error, $v_t, \ldots, v_n$ is the part after the error, and p is the error symbol. The difference between $\bar{a}$ and a is calculated as:

$$\bar{a} - a \equiv \sum_{i=j}^{n} v_i + j \cdot p \pmod{2n+1}. \quad (3)$$

  This difference is then used to correct the error.

- If $\bar{a} - a \leq \sum_{i=1}^{N} r_i$, we deduce that $p = 0$, and this insertion occurs before the $(\bar{a} - a)$-th "1" in the sequence, counted from back to front. Otherwise, the insertion is "1", and its position is after the $(\bar{a} - a - \sum_{i=1}^{N} r_i + 1)$-th "0" in the sequence. The handling of substitution and deletion errors is similar to that of insertion errors.

SISO decoder

In the SISO algorithm, the core idea is inspired by bit-by-bit optimal decoding methods based on concatenated structure codes (e.g., BCJR algorithm). Based on the bit-level MAP criterion, the goal is to calculate the posterior probability (APP) of a specific bit $v = (v_1, v_2, \ldots, v_n)$ being 0 or 1, given the received word $r = (r_1, r_2, \ldots, r_N)$. Specifically, the state S_t is defined as the aggregate value at time t, representing the dictionary state and the drift value caused by deletion and insertion errors, while D_t denotes the drift value, and d_t represents the difference between insertion and deletion errors.

The SISO decoder decodes both single and multiple errors, the basic steps are as follows:

- Given the received word and initial state, the MAP decoding criterion is represented by the factor graph as follows:

$$Pr[v_1^n|r_1^N, S_0 = s_0, D_0 = d_0]Pr[r_1^N|S_0 = s_0, D_0 = d_0]$$
$$= \prod_{t=1}^{n} \mathcal{F}[v_t, r_{d_{t-1}+t}^{d_t+t}, s_t, d_t], \quad (4)$$

  where each factor is

$$\mathcal{F}[v_t, r_{d_{t-1}+t}^{d_t+t}, s_t, d_t] \quad (5)$$
$$= Pr[v_t]Pr[r_{d_{t-1}+t}^{d_t+t}|v_t, s_t, s_{t-1}, d_t, d_{t-1}]Pr[s_t, d_t|s_{t-1}, d_{t-1}].$$

This factorization is represented by a factor graph, where each factor corresponds to a node in the graph.

- To compute the posterior probability of the bit v_t, a SISO decoder based on the BCJR algorithm is used. Given the received word r, the log-posterior probability (log-APP) of the bit being 0 or 1 can be computed by the following formula:

$$L(v_t) = \log \frac{Pr[v_t = 0|r, \mathcal{S}]}{Pr[v_t = 1|r, \mathcal{S}]}$$
$$= \log \frac{Pr[r, \mathcal{S}|v_t = 0]}{Pr[r, \mathcal{S}|v_t = 1]} + \log \frac{Pr[v_t = 0]}{Pr[v_t = 1]}. \quad (6)$$

  where $\mathcal{S}$ represents the aggregated state at each time step. Next, when calculating $Pr[r, \mathcal{S}|v_t = b]$, the following decomposition can be used:

$$Pr[r, \mathcal{S}|v_t = b] =$$
$$\sum_{(d',d)\in\mathcal{D}_t,(s',s)\in\sum_t^b} \alpha_{t-1}(s', d') \cdot \gamma_t(s', d', s, d) \cdot \beta_t(s, d). \quad (7)$$

  where $\alpha_{t-1}(s', d') = Pr[r_1^{d'+t-1}, s', d']$ is the message computed by the "forward recursion"; $\beta_t(s, d) = Pr[r_{d+t+1}^N|s, d]$ is the message computed by the "backward recursion"; $\gamma_t(s', d', s, d)$ is the probability computed based on the relationships between the received word, dictionary state, and drift state.

- To compute the above probabilities, $\alpha_t(s, d)$ and $\beta_t(s, d)$ can be obtained through the following recursion relations:

$$\alpha_t(s, d) = \sum_{(s',s)\in\sum_t,(d',d)\in\mathcal{D}_t} \gamma_t(s', d', s, d) \cdot \alpha_{t-1}(s', d').$$
$$\beta_t(s', d') = \sum_{(s',s)\in\sum_{t+1},(d',d)\in\mathcal{D}_{t+1}} \gamma_{t+1}(s', d', s, d) \cdot \beta_{t+1}(s, d). \quad (8)$$

  where $\sum_t$ represents all possible dictionary states.

- Using the above recursion relations and transition formulas, the state transition probability $\gamma_t(s', d', s, d)$ at each time step can be calculated using the following formula:

$$\gamma_t(s', d', s, d) = Pr[r_{d'+t}^{d+t}, s, d|s', d']$$
$$= Pr[s|s'] \cdot Pr[r_{d'+t}^{d+t}, d|s', d', s]. \quad (9)$$

  where $Pr[s|s']$ represents the conditional probability between dictionary states, and $Pr[r_{d'+t}^{d+t}, d|s', d', s]$ is the conditional probability of the received word and drift state.

- Once the log-posterior probability (log-APP) of the bit is computed, a hard decision can be made based on the sign of $L(v_t)$:

$$\hat{v}_t = \text{sign}(L(v_t)). \quad (10)$$

Then, the original information bits can be recovered through the inverse operation.

The above methods provide the main model architectures for traditional HD and SISO decoding algorithms, laying the theoretical foundation for our research.

METHOD

In this section, the overall framework, the symbol- and statistic-based codeword embedding, and the masking strategy of the attention mechanism are introduced.

Framework

As shown in figure 1, the TVTD adopts a simplified transformer-based sequence-to-sequence (seq2seq) architecture [17]. The encoder of the seq2seq model is reduced to the embedding of the corrupted codeword, enhancing computational efficiency. Such embedding comprises two parts: the symbol embedding and the statistic embedding of the corrupted codeword. The decoder follows a next-symbol prediction scheme, which predicts the transmitted/groundtruth codeword bit by bit using multi-layered attention mechanisms with the corrupted codeword embedding as memory. In this process, the standard upper triangular mask for next-symbol prediction is combined with a window mask, further reducing the method's complexity.

During training, the teacher forcing strategy is employed, where the masked and shifted groundtruth codeword is used to predict the next symbol. The model is optimized using a cross-entropy loss function. In the testing phase, the model generates the codeword autonomously, using predictions from previous iterations as input.

Symbol and statistic-based codeword embedding

The embedding process for codewords is discussed. It is observed that using only the symbol embedding of a codeword results in poor correction accuracy. Motivated by the HD decoder, the statistics of the symbol positions are integrated into the codeword embedding in the proposed method. Both received/corrupted codewords and transmitted/groundtruth codewords are embedded in the same manner. For each VT codeword, we perform the following operations to help the model capture its feature information.

Let $\boldsymbol{c} = (c_1, c_2, \ldots, c_n), c_i \in \{0, 1\}$ be a codeword of length n. The codeword embedding

$$\Phi(\boldsymbol{c}) = \text{concat}(\Phi_{\text{stat}}(\boldsymbol{c}), \Phi_{\text{sym}}(\boldsymbol{c})) \qquad (11)$$

is a concatenation of the symbol embedding $\Phi_{\text{sym}}(\boldsymbol{c})$ and the statistic embedding $\Phi_{\text{stat}}(\boldsymbol{c})$.
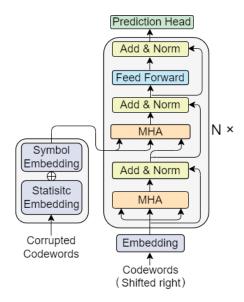

FIG. 1. Framework of the TVTD architecture. A seq2seq model is employed, where the encoder is simplified to the codeword embedding of the corrupted codeword. The decoder, on the other hand, adopts a next-symbol prediction scheme enhanced by a combined masking strategy.

Symbol embedding of a codeword

In general, the embedding of sequences in a seq2seq model consists of word embeddings and positional embeddings, which are typically added or concatenated. The positional embedding encodes the positional information of the words and can be either fixed or learnable.

In this work, due to the simplicity of the binary symbols in the codeword, the learnable positional embedding is integrated into the symbol embedding. Specifically, two learnable embedding vectors are associated with each position of the codeword, corresponding to symbol 0 and symbol 1 at that position. The symbol embedding $\Phi_{\text{sym}}(\boldsymbol{c}) = (\phi_1, \phi_2, \ldots, \phi_n)$ for codeword $\boldsymbol{c}$ is formulated as

$$\phi_i = (1 - c_i) * \boldsymbol{e}_{i0} + c_i * \boldsymbol{e}_{i1}, \qquad (12)$$

where $\boldsymbol{e}_{i0}$ and $\boldsymbol{e}_{i1}$ are the learnable embedding vectors associated with position i. An example is illustrated in fig. 2, where the embedding vectors are gathered based on the symbols in the codeword.
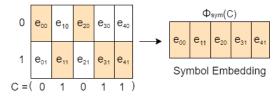

FIG. 2. Symbol embedding of a codeword. For a codeword $\boldsymbol{c} = (0, 1, 0, 1, 1)$, the vectors $(\boldsymbol{e}_{00}, \boldsymbol{e}_{11}, \boldsymbol{e}_{20}, \boldsymbol{e}_{31}, \boldsymbol{e}_{41})$ are gathered to form the symbol embedding of the codeword.

Statistic embedding of a codeword

The VT code corrects IDS errors using redundancy information based on joint symbol-position constraints, as described in eq. (1) and section VT encoder . Motivated by this, the joint statistics of position and symbol are embedded as part of the final codeword embedding.

Mimicking eq. (2), the two statistics that sum the position indices of symbol 0s and the position indices of symbol 1s, respectively, are computed as

$$s_0 = \sum_{i=1}^{n} i \cdot \mathbb{1}_{\{c_i=0\}}, \qquad (13)$$

$$s_1 = \sum_{i=1}^{n} i \cdot \mathbb{1}_{\{c_i=1\}}, \qquad (14)$$

where $\mathbb{1}_X$ denotes the indicator function of set X.

Two lookup tables ϕ_0, ϕ_1 are used to store the embedding vectors for the possible values of the two statistics, respectively. The statistic embedding of the codeword is obtained by retrieving the embedding vectors $\Phi_{\text{stat}} = (\phi_0(s_0), \phi_1(s_1))$ from the tables according to s_0 and s_1, respectively.

Combined masking strategy

An upper triangular mask is commonly used in the training phase of a transformer in a seq2seq task to prevent the model from "seeing" the future context to be predicted. To improve the efficiency of the proposed model, a combined mask, consisting of both the triangular mask and a window mask, is employed during training. In this setup, the model predicts the next symbol based on the corrupted codeword and the sliding window segment of the predicted codeword.

Upper triangular masking

The upper triangular mask M_{upper} has a size of $L \times L$, where L is the length of the codeword embedding. The mask is defined as

$$M_{\text{upper}}(k, j) = \begin{cases} 0, & \text{if } k \leq j, \\ -\infty, & \text{if } k > j, \end{cases} \qquad (15)$$

where the k and j represent the row and column indices of the attention matrix, respectively.

Window masking

The concept of performing attention within a local window was first introduced in the Swin Transformer [26], which was designed as a backbone for computer vision tasks.

In this work, the window mask M_{window}, with a size of $L \times L$, restricts each position's attention to a local context within a window size w. Its definition is as follows:

$$M_{\text{window}}(k, j) = \begin{cases} 0, & \text{if } |k - j| \leq w, \\ -\infty, & \text{otherwise.} \end{cases} \qquad (16)$$

By controlling the window size w, the information used to predict the next symbol of the groundtruth codeword can be varied, achieving a balance between time consumption and model performance.

Combining the masks

During the training phase, the combined mask applied in the attention calculation is the sum of the upper triangular mask and the window mask. Specifically, the attention is calculated as

$$A_H(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d}} + M \right) V, \qquad (17)$$

where the mask $M = M_{\text{upper}} + M_{\text{window}}$, and the $(Q, K, V) \in \mathbb{R}^{L \times d}$ are the query, key, and value matrices of the attention mechanism [17].

During the testing phase, the local segment $c[-w :]$ of the codeword is used, along with the corrupted codeword embedding, to predict the next symbol in each iteration.

EXPERIMENTAL EVALUATION

Experimental setup

To ensure that the experimental results are widely representative, experiments were conducted with codewords of three lengths: 20-bit (short codewords), 68-bit (medium codewords), and 120-bit (long codewords). The learning rate was set to 0.0001 and adjusted using a cosine annealing scheduler [27]. The model was trained for 200 epochs. The embedding dimension was set to 512, with the feed-forward network dimension being four times the embedding dimension [28]. The decoder consists of $N = 3$ transformer layers, each with 8 attention heads. A quarter of the codeword length was chosen as the window size for medium and long codewords, while the window size is 20 for short codewords. The source code is uploaded in the Supplementary Material, and will be made publicly accessible upon the manuscript's publication.

For the experiments with 20-bit codewords, 80% of the codewords and their corrupted counterparts were selected as the training set, while the remaining 20% were used as the testing set. For medium and long codewords, since

the number of codewords is numerous, $240,000$ (resp. $60,000$) random codewords were used for training (resp. testing).

Metrics

To comprehensively evaluate the performance of the proposed method, both the Bit Error Rate (BER) and Frame Error Rate (FER) are employed. The BER represents the fraction of erroneous bits in the decoded codewords, while the FER is the ratio of codewords corrected in error to the total number of corrupted codewords received. It is worth noting that since the codewords are binary and randomly generated, the BER cannot exceed 50%.

Comparison experiments

Experiments were conducted across two methods, including HD and SISO, for comparison. Metrics were collected under two tasks: correcting a fixed number of errors and correcting errors independently introduced by a fixed error rate.

Results on correcting fixed number of errors

To evaluate the performance of the compared approaches in correcting multiple errors, fixed numbers of 1, 2, 3, or 4 IDS errors were introduced to each codeword before correction, with each type of IDS error randomly distributed. For each test involving n IDS errors of TVTD, the model was trained to correct a random number of errors ranging from 0 to n, and tested to correct n IDS errors. The experimental results are presented in table I.

As shown in table I, all three compared approaches demonstrate satisfactory performance when correcting a single IDS error. This is expected, as the VT code is specifically designed for single IDS error correction. However, when more than one error is introduced, the performance of HD and SISO declines sharply, while the proposed TVTD maintains commendable accuracy, outperforming the other methods by a significant margin. The FER of TVTD on multiple errors suggests that the VT code for correcting IDS errors may not be 'optimal', nor will 1-FER should be no more than 50%.

It was also observed that a higher number of errors leads to higher error rates, as expected. Furthermore, when the length of the code increases, the performance degradation is mitigated as the number of errors grows. It is conjectured that the ratio of errors in the codeword is more closely related to the final performance, as a longer code length results in a lower proportion of errors.

Results on independently distributed errors

In some applications like DNA storage, IDS errors typically occur independently within the sequence, posing a challenge for decoding methods. To evaluate the performance of the proposed TVTD under such conditions, experiments were conducted by introducing independent IDS errors to the codeword. The error rates were set to 1%, 3%, and 5%, with each error type having equal probability. The experimental results are presented in table II.

For short codewords like $VT(20, 14)$, the likelihood of encountering multiple IDS errors in a codeword is much lower compared to the other two codes. All three decoders effectively correct single IDS errors, as shown in table I. As a result, the three approaches exhibit similar performance on $VT(20, 14)$. On the other hand, since multiple errors are likely to occur in a sequence of length 68 or 120, the proposed method outperforms HD and SISO by a large margin when decoding $VT(68, 60)$ and $VT(120, 112)$. Specifically, when the channel error rate reaches 5%, HD and SISO fail to decode meaningful information, while the TVTD still achieves more than 98% bit accuracy and 70% frame accuracy.

Ablation study

The ablation study and hyperparameter optimization were also conducted to evaluate the effectiveness of the proposed methods. All the experiments in the ablation study are performed on correcting codes from $VT(68, 60)$.

Window size

The window size w controls the size of the segment from which the model predicts the next symbol. It also affects the time consumption of the method. The BERs and FERs were recorded with different choices of window size w on correcting corrupted codewords from $VT(68, 60)$ with fixed number of errors. The results are illustrated in table IV[29].

As suggested by table IV, the accuracies improve as the window size increases. For correcting 2 and 3 errors, the performance reaches a plateau after $w = 16$. Meanwhile, for correcting 4 errors, the plateau point occurs later, indicating that correcting multiple errors is more closely related to global dependencies within the sequence.

Statistic embedding

The statistic embedding of the codeword, introduced as part of the codeword embedding in section Symbol and

	# Errors	VT(20, 14)				VT(68, 60)				VT(120, 112)			
		1	2	3	4	1	2	3	4	1	2	3	4
HD	1-BER (%)	**100.0**	67.9	65.3	60.7	**100.0**	69.0	65.9	61.6	**100.0**	69.2	65.4	60.7
	1-FER (%)	**100.0**	9.0	8.9	2.5	**100.0**	3.1	2.6	0.2	**100.0**	2.6	2.2	0.0
SISO	1-BER (%)	99.9	82.1	75.4	**72.4**	100.0	86.3	83.6	76.5	**100.0**	85.4	79.7	77.6
	1-FER (%)	97.9	13.1	5.0	1.2	99.6	4.6	1.4	0.1	**100.0**	1.7	0.1	0.0
TVTD	1-BER (%)	**100.0**	**83.7**	**76.5**	71.9	99.9	**98.3**	**98.2**	**97.0**	100.0	**99.8**	**99.5**	**99.3**
	1-FER (%)	99.9	**32.5**	**13.6**	**6.4**	99.0	**88.0**	**76.7**	**63.3**	99.3	**91.7**	**83.0**	**75.3**

TABLE I. Performance of different VT decoders: HD, SISO, and TVTD, at various numbers of errors. The VT codes with different code/message lengths VT(20, 14), VT(68, 60), and VT(120, 112) were tested.

	Error Rate	VT(20, 14)			VT(68, 60)			VT(120, 112)		
		1%	3%	5%	1%	3%	5%	1%	3%	5%
HD	1-BER (%)	99.6	97.0	93.7	96.5	85.3	73.6	92.4	72.9	61.7
	1-FER (%)	**98.9**	**90.3**	**80.9**	89.0	52.1	22.2	73.4	20.0	4.8
SISO	1-BER (%)	99.7	**97.7**	**95.2**	97.7	87.9	83.9	94.9	80.0	71.6
	1-FER (%)	98.1	89.1	77.4	85.5	33.9	10.5	70.1	6.0	0.3
TVTD	1-BER (%)	**99.8**	97.7	94.8	**99.8**	**98.9**	**98.2**	**99.9**	**99.3**	**98.6**
	1-FER (%)	**98.9**	**90.3**	79.1	**97.5**	**86.1**	**70.3**	**96.9**	**81.0**	**67.4**

TABLE II. Performance of different VT decoders: HD, SISO, and TVTD, at various error rates. The VT codes with different code/message lengths VT(20, 14), VT(68, 60), and VT(120, 112) were tested.

statistic-based codeword embedding , is evaluated in ablation studies, as shown in table V[30]. In this table, the column headers like "w/w" indicate whether the statistic embedding is applied to the embedding of the corrupted codeword and the predicted codeword. For instance, "wo/w" means that the embedding of the corrupted codeword is performed **with**o**ut** the statistic embedding, while the model is trained and tested to predict the codeword **w**ith the statistic embedding. Section Symbol and statistic-based codeword embedding suggests that introducing the statistic embedding into the input codeword embedding significantly improves the frame accuracies. Meanwhile, in the case of "wo/wo", where the statistic embedding is absent, the model's performance degenerates by a large margin.

Different encoder of the seq2seq architecture

A commonly used seq2seq model consists of the encoder and decoder, which are both stacks of transformer layers. In this work, direct embedding of the corrupted codeword is engaged for the cross-attentions to predict the groundtruth codeword. Experiments were conducted to find out whether such a setting declines the performance while improving the complexity efficiency.

Let TVTD(0+3) be the proposed default architecture which uses codeword embedding directly as the encoder memory and 3-layered decoder to predict the groundtruth codeword, while TVTD(0+3) denotes the seq2seq model with 3-layered encoder and 3-layered de-

coder. The results are illustrated in table III. It is suggested that using more complex architecture results in slight performance improvement. However, it is also observed that the model slows down by about 44% to 66% comparing to the simple model.

Complexity

In the TVTD model, the key sources of computational complexity come from the cross-attention, self-attention, and feedforward networks. The complexity of the cross-attention is $O(\ell^2 d)$, where ℓ is the length of the target and source sequences, and d is the embedding dimension. The complexity of the self-attention is $O(\ell w d)$, where w is the window size, typically smaller than the target sequence length ℓ. The feedforward network's computational complexity is $O(\ell d^2)$, indicating the computation cost for each target position. Overall, the complexity of the decoder can be expressed as $O(\ell^2 d + \ell w d + \ell d^2)$. Therefore, the computational complexity of the model is mainly influenced by the target sequence length, source sequence length, and embedding dimension, with the cross-attention computation being the primary bottleneck.

Although the typical transformer has a complexity of $O(\ell^2)$, the TVTD still benefits from parallel computation on a GPU. The time consumed for correcting $10,000$ corrupted codewords with 2 errors was recorded. The results are as follows: HD (Xeon): 7.6s @ VT(68, 60); SISO (Xeon): 15747.4s @ VT(20, 14); and TVTD (NVIDIA

	# Errors	VT(20,14)				VT(68,60)				VT(120,112)			
		1	2	3	4	1	2	3	4	1	2	3	4
TVTD(3+3)	1-BER(%)	100.0	83.9	76.3	72.2	99.9	99.3	98.6	97.7	99.9	99.8	99.7	99.4
	1-FER(%)	100.0	34.5	15.8	7.1	99.7	90.5	80.5	67.8	99.0	93.0	87.0	77.3
TVTD(0+3)	1-BER (%)	100.0	83.7	76.5	71.9	99.9	98.3	98.2	97.0	100.0	99.8	99.5	99.3
	1-FER (%)	99.9	32.5	13.6	6.4	99.0	88.0	76.7	63.3	99.3	91.7	83.0	75.3

TABLE III. Performance of TVTD with different seq2seq encoder architectures. TVTD(3+3) uses a 3-layered transformer as the seq2seq encoder, while TVTD(0+3) is the default setup using direct embedding as the encoder.

# Error	Metric	$w=1$	$w=2$	$w=4$	$w=8$	$w=16$	$w=32$	all
1	1-BER (%)	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	1-FER (%)	100.0	100.0	99.8	100.0	100.0	100.0	100.0
2	1-BER (%)	99.1	99.1	99.2	99.2	99.2	99.2	99.2
	1-FER (%)	78.7	81.4	84.2	85.4	87.0	86.9	87.4
3	1-BER (%)	97.9	97.8	98.0	98.1	98.2	98.2	98.2
	1-FER (%)	57.4	60.4	66.9	69.5	73.8	74.7	74.7
4	1-BER (%)	96.5	96.6	96.5	96.6	96.7	96.9	96.9
	1-FER (%)	35.2	43.7	44.1	47.8	51.5	61.4	61.5

TABLE IV. Performance of TVTD correcting fixed number of errors with various window sizes w ranging from 1 to the whole the sequence.

# Error	Metric	w/w	w/wo	wo/w	wo/wo
1	1-BER (%)	100.0	100.0	99.9	99.7
	1-FER (%)	100.0	100.0	97.3	89.4
2	1-BER (%)	99.2	99.2	98.8	98.7
	1-FER (%)	87.7	87.3	73.5	73.1
3	1-BER (%)	98.2	98.2	98.1	97.7
	1-FER (%)	73.4	72.9	71.2	62.2
4	1-BER (%)	96.8	96.8	96.9	96.6
	1-FER (%)	51.6	52.3	57.2	47.4

TABLE V. Performance metrics for different methods at various error rates across different VT codes.

A40): 80.7s @ VT(68, 60). These results suggest that TVTD consumes significantly less time than the other soft decoder, SISO, and requires about 10 times more time than HD to achieve the capability of multiple error correction.

CONCLUSION

In this paper, a transformer-based binary VT code decoder was proposed, marking the first neural network implementation of a VT decoder. Within the TVTD framework, symbol- and statistic-based codeword embeddings, and a combined masking strategy were introduced to enhance model performance and efficiency. Experimental results demonstrated that TVTD significantly outperforms HD and SISO, particularly in its capacity to correct multiple errors, which beyonds the original design scope of VT codes. Abaltion studies further validated the effectiveness of the proposed techniques.

[*] yaliwei222¨@tju.edu.cn
[†] jiaxiang.guo@tju.edu.cn
[‡] daiyufan@tju.edu.cn
[§] yanzh@tju.edu.cn

[1] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. LeProust, B. Sipos, and E. Birney, Towards practical, high-capacity, low-maintenance information storage in synthesized dna, Nature **494**, 77 (2013).

[2] Y. Erlich and D. Zielinski, Dna fountain enables a robust and efficient storage architecture, Science **355**, 950 (2017).

[3] M. A. Sini and E. Yaakobi, Reconstruction of sequences in dna storage, in *2019 IEEE International Symposium on Information Theory (ISIT)* (IEEE, 2019) pp. 290–294.

[4] L. Song, F. Geng, Z.-Y. Gong, X. Chen, J. Tang, C. Gong, L. Zhou, R. Xia, M.-Z. Han, J.-Y. Xu, B.-Z. Li, and Y.-J. Yuan, Robust data storage in dna by de bruijn graph-based de novo strand assembly, Nature Communications **13**, 5361 (2022).

[5] S. M. H. T. Yazdi, H. M. Kiah, E. Garcia-Ruiz, J. Ma, H. Zhao, and O. Milenkovic, Dna-based storage: Trends and methods, IEEE Transactions on Molecular, Biological and Multi-Scale Communications **1**, 230 (2015).

[6] R. Heckel, G. Mikutis, and R. N. Grass, A characterization of the dna data storage channel, Scientific Reports **9**, 1 (2019).

[7] X. Li, M. Chen, and H. Wu, Multiple errors correction for position-limited dna sequences with gc balance and no homopolymer for dna-based data storage, Briefings in Bioinformatics **24**, bbac484 (2023).

[8] 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 , 1 (2022).

[9] 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.

[10] W. Press, J. Hawkins, S. Jones, J. Schaub, and I. Finkelstein, Hedges error-correcting code for dna storage corrects indels and allows sequence constraints, Proceedings of the National Academy of Sciences **117**, 18489 (2020).

[11] M. Davey and D. Mackay, Reliable communication over channels with insertions, deletions, and substitutions, IEEE Transactions on Information Theory **47**, 687 (2001).

[12] V. Buttigieg, S. Wesemeyer, and J. Briffa, Time-varying block codes for synchronisation errors: maximum a posteriori decoder and practical issues, The Journal of Engineering **2014**, 1 (2014).

[13] V. I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, Soviet physics. Doklady **10**, 707 (1965).

[14] R. Varšamov and G. Tenengolts, A code which corrects single asymmetric errors, Avtomat. i Telemeh **26**, 4 (1965).

[15] K. Cai, Y. M. Chee, R. Gabrys, H. M. Kiah, and T. T. Nguyen, Correcting a single indel/edit for dna-based data storage: Linear-time encoders and order-optimality, IEEE Transactions on Information Theory **67**, 3438 (2021).

[16] Z. Yan, G. Qu, and H. Wu, A novel soft-in soft-out decoding algorithm for vt codes on multiple received dna strands, in *2023 IEEE International Symposium on Information Theory (ISIT)* (IEEE, Taipei, Taiwan, 2023) pp. 1–6.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Łukasz Kaiser, and I. Polosukhin, Attention is all you need, in *Advances in Neural Information Processing Systems* (2017) pp. 5998–6008.

[18] H. Kim, Error correction codes, in *Wireless Communications Systems Design* (Wiley Telecom, 2015) Chap. 139, pp. 123–207.

[19] E. Nachmani and L. Wolf, Hyper-graph-network decoders for block codes, in *Advances in Neural Information Processing Systems* (2019) pp. 2326–2336.

[20] L. Lugosch and W. J. Gross, Neural offset min-sum decoding, in *2017 IEEE International Symposium on Information Theory (ISIT)* (IEEE, 2017) pp. 1361–1365.

[21] 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)* (IEEE, 2016) pp. 341–346.

[22] Y. Choukroun and L. Wolf, Error correction code transformer, in *Advances in Neural Information Processing Systems 35 (NeurIPS 2022) Main Conference Track* (2022).

[23] Y. Choukroun and L. Wolf, A foundation model for error correction codes, in *Proceedings of the 2024 International Conference on Learning Representations (ICLR 2024)* (2024) submitted to ICLR 2024.

[24] Y. Choukroun and L. Wolf, Learning linear block error correction codes, in *Proceedings of the 41st International Conference on Machine Learning*, PMLR, Vol. 235 (2024) pp. 8801–8814.

[25] K. Saowapa, H. Kaneko, and E. Fujiwara, Systematic deletion/insertion error correcting codes with random error correction capability, in *Proceedings 1999 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (EFT'99)* (1999) pp. 284–292.

[26] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, and Z. Zhang, Swin transformer: Hierarchical vision transformer using shifted windows, in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)* (IEEE, 2021) pp. 10012–10022.

[27] I. Loshchilov and F. Hutter, SGDR: Stochastic gradient descent with warm restarts, in *International Conference on Learning Representations* (2017).

[28] R. Xiong, Y. Yang, D. He, K. Zheng, S. Zheng, C. Xing, H. Zhang, Y. Lan, L. Wang, and T.-Y. Liu, On layer normalization in the transformer architecture, arXiv preprint arXiv:2002.04745 (2020).

[29] This ablation study used smaller training and testing sets, consisting of $160,000$ and $40,000$ samples, respectively.

[30] This ablation study used smaller training and testing sets, consisting of $160,000$ and $40,000$ samples, respectively.