Supplementary Information for *Machine Learning Discovers New Champion Codes*

1 Generalised Toric Codes

The most common (error-correcting) codes are *linear codes* and defined over subspace C of the vector space $\mathbb{F}_q^n$. Mathematically, a linear code is a k-dimensional linear subspace C of the vector space $\mathbb{F}_q^n$ over a finite field with q elements. The vectors in C are *codewords* of block length n, and *size* is the number of possible codewords, which is equal to q^k. The *weight* of a codeword is the number of its non-zero elements (vector components), and the *distance* between two codewords is the Hamming distance between them, that is, the number of elements in which they differ. The distance d of the linear code is the minimum weight of its nonzero codewords or, equivalently, the minimum distance between distinct codewords. A linear code C has three main parameters – length n, dimension k, and distance d, all together denoted as $[n, k, d]_q$. A general reference for coding theory is [1], and one can find theoretical upper and lower bounds on the highest minimum distance for a given n and k with the https://codetables.de/ [2].

A linear code can be defined as linear subspace C with dimension k of the vector space $\mathbb{F}_q^n$. Basis codewords of this space can be written as generator matrix G. The parity-check matrix P of the code is defined as a matrix P having C as its null space. We say that V is the set of vertices of the generalised toric code C_V.

Minimum distance d_V is a measure of the accuracy of the code for error correction. It is equal to the minimum of the Hamming distance between any two codewords, while Hamming distance defined as a number of unequal elements between two vectors. An important task is to seek codes with a minimum Hamming distance d that is as large as possible for a given n and k; this is sometimes referred to as "the main coding theory problem" [1]. In particular, we refer to any code with a larger minimum distance than the currently known lower bounds as **champion code**, as in [3].

To calculate this minimum distance, we use an algorithm known as the Brouwer-Zimmermann (BZ) algorithm [4], also described in [5], which we explain briefly. The algorithm generates several equivalent generator matrices, each with pivots located in a unique set of columns (known as the information set). The relative rank of a generator matrix is defined as the number of pivot positions in its information set that are independent of previously constructed matrices.

The algorithm involves enumerating all possible combinations of r generators for increasing values of r. For each r, if r becomes larger than the difference between the matrix's actual rank (i.e. its dimension) and its relative rank, the algorithm increases the lower bound on the minimum weight by one. Meanwhile, the upper bound on the minimum weight is determined by finding the smallest weight among the enumerated codewords. The computation completes once the lower and upper bounds converge. Despite several improvement by [5], the BZ algorithm is computationally costly – it has polynomial in k and exponential in q complexity [6]. This plays a key role over $\mathbb{F}_8$. We can characterise the code $C_V(\mathbb{F}_q)$ with two matrices, the generator matrix $G_V(\mathbb{F}_q)$ and its parity-check matrix $P_V(\mathbb{F}_q)$, defined as follows:

Definition 1. *Suppose C is a linear code with block length n and dimension k.*

- *The generator matrix G of C is a $k \times n$ matrix whose rows form a basis for the code, such that any codeword c is in the row-span and thus can be expressed as $c = m \cdot G$ for some length-k vector m.*
- *The canonical form for the generator matrix G is $G = [I_k|H]$, where I_k is the $k \times k$ identity matrix and H is a $k \times (n-k)$ matrix.*
- *The dual generator matrix $G^{\perp}$ of G is any matrix whose rows form a basis for the dual code $C^{\perp}$, where*

$$C^{\perp} = \{y \in \mathbb{F}_q^n : y \cdot c = 0 \text{ for all } c \in C\}.$$

  The parity check matrix P of G is then an $(n-k) \times n$ matrix satisfying $GP^T = 0$. Because P satisfies $GP^T = 0$ and its rows generate $C^{\perp}$, P is typically considered a dual generator matrix.

Likewise, given the canonical form of the generator matrix, the canonical form of the parity-check matrix P is $[-H^T|I_{n-k}]$. Thus, there is no ambiguity when considering the equivalent generator and parity-check matrices of a linear code C. Hereafter, we assume the canonical form when discussing the generator and parity-check matrices.

Toric codes are a class of codes introduced by J. Hansen in [7], which serves as a valuable source for constructing linear codes. As we will see, toric codes (we will refer to them as non-generalised to avoid any ambiguity) and generalised toric codes are defined in the same terms with the only difference being that the non-generalised toric codes are built from a set of vertices belonging to a certain lattice polytope, while generalised toric codes can be built from any set of vertices on a lattice [1]. Therefore, the set of non-generalised toric codes is a subset of the set of generalised toric codes. As we will see later, working with generalised toric codes is easier in our case and, at the same time, allows us to infer knowledge about toric codes.

It is important to emphasise that the term *toric codes* was also used in quantum computing by Kitaev [9]. However, this is entirely different. Moreover, while Hansen defined toric codes using toric varieties and polytopes, they

[1]See [8] for more details.

can also be described within the framework of evaluation codes. We need not delve into the details of defining toric codes here because we will now introduce the generalized version by J. Little [10], which have proven useful in identifying champion codes [11]. In [3], G. Brown and A. Kasprzyk exhaustively classified all generalised toric codes over fields from $\mathbb{F}_3$ to $\mathbb{F}_7$ (and $\mathbb{F}_8$ partially), identifying seven champion linear codes that outperformed those listed in the code tables at the time. Although their method is exhaustive, it is computationally expensive because of the vast number of possible generalised toric codes, making a systematic search infeasible for larger values of q. Upon further investigation, the calculation of the minimum Hamming distance for any linear code is another bottleneck for such a search.

In this study, we focused on generalised toric codes. This comes at the cost of losing certain geometric results specific to toric codes such as the Minkowski length [12]. We will address this trade-off in detail later; for now, we begin by introducing generalised toric codes.

Consider the planar lattice grid $[0, q-2]^2$ over the field $\mathbb{F}_q$ for a prime power q and primitive element $\xi \in \mathbb{F}_q$ (i.e. the primitive $(q-1)$-th root of unity in the multiplicative subgroup $\mathbb{F}_q^*$). For all $0 \leq i, j \leq q-2$, we define $P_{i,j} = (\xi^i, \xi^j) \in (\mathbb{F}_q^*)^2$, and for each $u = (u_1, u_2) \in [0, q-2]^2$, we define the mapping

$$\begin{aligned} &e(u) : \mathbb{F}_q^* \times \mathbb{F}_q^* \longrightarrow \mathbb{F}_q \\ &P_{i,j} = (\xi^i, \xi^j) \longrightarrow e(u)(P_{i,j}) = (\xi^i)^{u_1}(\xi^j)^{u_2}. \end{aligned} \tag{1}$$

Then, we have

Definition 2. *Let $V = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ be a set of vertices from $[0, q-2]^2$. We define the corresponding generalised toric code $C_V(\mathbb{F}_q)$ over the field $\mathbb{F}_q$ associated to V as the the linear code of block length $n = (q-1)^2$ and dimension $k = \dim C_V(\mathbb{F}_q)$ spanned by the vectors in*

$$\left\{ (e(u)(P_{i,j}))_{0 \leq i,j \leq q-2} = (\xi^{iu_1 + ju_2})_{0 \leq i,j \leq q-2} \ : \ u \in V \right\}. \tag{2}$$

Note that the dimension k of our code may differ from the number of vertices m. As with any linear code, the minimum Hamming distance $d = d_V(\mathbb{F}_q)$ of $C_V(\mathbb{F}_q)$ is then defined as the minimum distance between any two codewords of $C_V(\mathbb{F}_q)$. We can then describe $C_V(\mathbb{F}_q)$ as an $[n, k, d]_q$-linear code. Sometimes, we also use m_V to reference the number of vertices m for code C_V.

Example: It is expedient to give an example here. If $q = 3$, the primitive element is $\xi = 2$ (as in $\mathbb{F}_3 \simeq \{0, 1, 2\}$, the multiplicative group is $\mathbb{F}_3^* \simeq \{1, 2\}$, and both are powers of 2 modulo 3), and we have a $[0, 1]^2$ lattice grid over $\mathbb{F}_3$.

Notice that:

$$\begin{aligned}
e(0,0)(P_{i,j}) &= 1\\
e(0,1)(P_{i,j}) &= 2^j\\
e(1,0)(P_{i,j}) &= 2^i\\
e(1,1)(P_{i,j}) &= 2^{i+j}\ .
\end{aligned} \tag{3}$$

Consider $V = \{(0,0),(0,1),(1,0),(1,1)\}$. Subsequently, the generalised code C_V is spanned by the following vectors:

$$(1,1,1,1),(1,1,2,2),(1,2,1,2),(1,2,2,1)$$

Then, the generator matrix G_V and parity-check matrix P_V of C_V are

$$G_V = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}, \quad P_V = \begin{pmatrix} 1 & 2 & 2 & 1 \end{pmatrix}$$

Note that the dimension of this code is not equal to the number of vertices.

By computation with a brute-force algorithm such as the Brouwer-Zimmermann (BZ) algorithm, we calculate that the minimum Hamming distance d_V is 2. Hence, C_V is a $[4,3,2]$-code.

Non-generalized toric codes are constructed by ensuring that the set of vertices forms a convex polytope over $[0,q-2]^2$.

If we consider non-generalised toric codes only, we gain access to geometric methods by connecting them to toric varieties, as described in [7]. In particular, many studies have used the theory of toric varieties, their cohomology, and intersection theory to obtain the upper and lower bounds for the minimum Hamming distance of the associated toric codes of certain shapes, as seen in [7], [12], and [13]. Although there are more tools to access by restricting to non-generalised toric codes, we decided to consider generalised toric codes instead for a few reasons:

1. Generating datasets for generalised toric codes is computationally cheaper compared to non-generalised toric codes.
2. As mentioned before, generalised toric codes have been used to discover champion codes. Although only one generalised toric code was required to discover a champion code over $\mathbb{F}_7$, as seen in [3], it is possible that more generalised toric codes are champion codes over $\mathbb{F}_8$.

We introduce a method to find champion codes over a given space, and apply this to the space of generalised toric codes.

2 Machine Learning Experiments for Generalised Toric Codes

This appendix expands the description of the machine learning experiments from main paper.

2.1 Datasets

In this section, we give more details on how the dataset were generated and prepared for training.

2.1.1 $\mathbb{F}_7$ dataset

To obtain a representative dataset, we generated a dataset of 100,000 tuples of the form (V, G_V, d_V) [2] for each number of vertices $m \in [5, 31]$. For a fixed m, we followed an algorithm 1. The resulting dataset is shown in Figure 1.

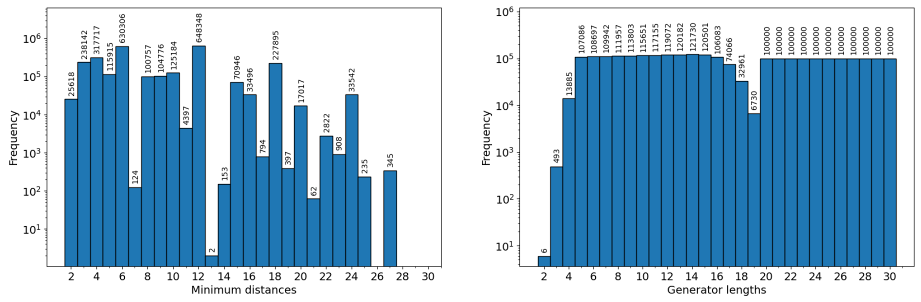

Figure 1: Histograms of the $\mathbb{F}_7$ codes dataset. The left histogram is the distribution of minimum distances, the middle one is the distribution of generators' dimensions, and the right is the distribution of dual generators' dimensions.

Algorithm 1 Generating datasets of size up to 100,000 over $\mathbb{F}_q$, for number of vertices m

```
Require: A positive integer $m \leq (q-1)^2$
Ensure: A dataset $S^*$ of 100,000 distinct tuples of the form $(V, G_V, P_V, d_V)$
    for $|V| = m$ vertices.
    1. Generate a set $S$ of 100,00 distinct uniformly random subsets of $[0, q-2]^2$
       of size $m$.
    2. Take $V \in S$, a set of $m$ vertices of $[0, q-2]^2$.
    3. Define $C_V$, using the order defined in 3.1.
    4. Calculate $G_V$ in standard form, and $d_V$.
    5. Return the tuple $(V, G_V, d_V)$ in a dataset, $S^*$.
    6. Repeat for each $V \in S$.
    7. Return $S^*$.
```

[2] The initial investigation showed that the generator matrices G_V and dual generator matrices P_V contain essentially the same information; therefore, we continued to generate datasets as a collection of tuples (V, G_V, d_V).

Note that the total number of subsets in our lattice grid is $2^{(q-1)^2}$ and the number of subsets of size m is $\binom{(q-1)^2}{m}$. Thus, only for $5 \leq m \leq 31$ there may exist more than 100,000 distinct codes over $\mathbb{F}_7$.

Generating the $\mathbb{F}_7$ dataset using Magma, which can perform commutations in parallel, on a computer cluster node with 96 Intel Xeon(R) Gold 6442Y processors (48 cores, 96 threads, 2.6-4.0 GHz) took approximately 24 h.

The resulting dataset of codes over $\mathbb{F}_7$ contains 2,700,000 examples. As illustrated in Figure 1, there is considerable variation in the frequencies associated with different minimum distances, with counts for individual minimum distances ranging from as few as two instances to over 600,000.

Training on such an unbalanced dataset requires additional steps beyond simply partitioning the data into train and test sets randomly. Specifically, it is important to split each subset of the dataset with the same minimum distance into train and test sets separately, merging them into full train and test sets with re-shuffling afterwards. Furthermore, class imbalance can be further mitigated through the application of oversampling/downsampling of the minority/majority classes. As will be described below, model trained for codes over $\mathbb{F}_7$ uses cross-entropy loss which allows also to down-weight the losses for oversampled and up-weight downsampled classes.

The final dataset for $\mathbb{F}_7$ codes contained around 550,000 and was used for training with 9:1 train/test split. The subsets with less than 500 elements were oversampled with replacement to 500 elements, and the subsets with more than 50,000 examples were downsampled to 50,000[3]. The only class with less than 10 elements, corresponding to a minimum distance 13, was used in the test set only.

It is important to note that the dimension of code C_V (the number of rows of the generator) can be equal to or smaller than the number of vertices from which the code is generated. This occurs when the rows of linear code are not linearly independent. Conversely, the dimension can be larger than the number of vertices for the dual codes.

2.1.2 $\mathbb{F}_8$ dataset

In the case of codes over $\mathbb{F}_8$, we were able to generate 100,000 codes only for the number of vertices 4-17, following an algorithm similar to 1. Due to the computation time required to calculate the minimum distances of codes of higher dimensions, we cut the number of generated codes for the number of vertices 17-39 to 1,000, with a limit on the computation time of 1 minute. An additional 20,000 codes were generated for the vertex numbers $m = 2, 3, 40-47$ afterwards. As can be seen in the figure 2[4], for $m = 2, 3$, the number of existing codes

[3]The goal was to get moderately imbalanced dataset with the smallest classes being no smaller than 1% of the biggest classes.

[4]The outlier at $m = 42$ is a result of a mistake when handling the data: 20,000 data points for $m = 41$ were added twice, and 20,000 for $m = 42$ was never added in the dataset. It was spotted after all the experiments were completed. The results are presented as is because it would be very time-consuming to repeat all the processes.

is smaller than 20,000 examples. A similar situation occurred with $m = 47$. The variation in the number of examples by minimum distance in the region $m = 17..40$ is caused by the limit on the minimum distance computation time we set in Magma, demonstrating the complexity of the procedure in this region. The resulting dataset contained 1,584,099 examples.

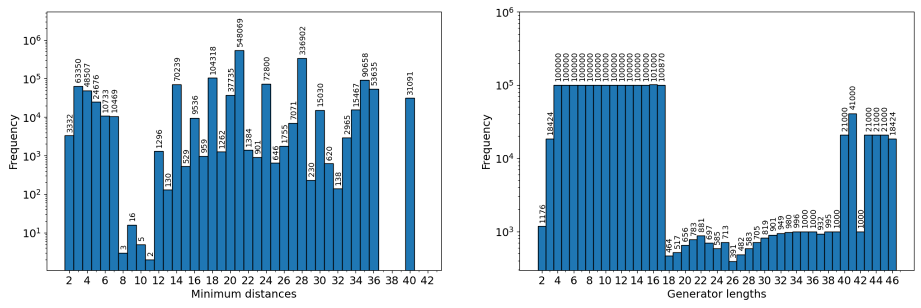

Figure 2: Histograms of the $\mathbb{F}_8$ codes dataset. The left histogram is the distribution of minimum distances, the right one is the distribution of generators' dimensions.

Generating the $\mathbb{F}_8$ dataset using Magma on a computer node with 96 Intel Xeon(R) Gold 6442Y processors (48 cores, 96 threads, 2.6 - 4.0 GHz) took around 48 hours.

An unexpected feature of the codes over $\mathbb{F}_8$, compared to $\mathbb{F}_7$ codes, is that dimensions of codes are always equal to the number of vertices from which these codes were generated.

The minimum distance prediction task for the codes over $\mathbb{F}_8$ is expected to be more complicated than for $\mathbb{F}_7$, but the amount of data we managed to collect was marginally smaller than for $\mathbb{F}_7$. To compensate for this data limitation, we developed with a two-stage training procedure. A pre-training stage was done on a larger dataset with approximately 300,000 examples, which included subsets (codes with the same minimum distance) with more than 10,000 examples; subsets with more than 20,000 examples were downsampled to 20,000. Then, in the training stage, all the subsets were used, with down-sampling or over-sampling to 600.

In goal was to train the model to extract useful features of codes on the common examples in the pre-training and then generalize this knowledge to all subsets in the training stage. In the later had a 10 times smaller learning rate to change the weights on a smaller scale than in pre-training to avoid losing any learned features.

The resulting performance is described is **Training and Performance** of the main paper. Here we will point out a couple of features found in the results.

Figure 3 reveals a notable trend: the model exhibits inferior performance for codes with minimum distances $d = 2, 14, 22$, and 27. However, a worse performance can be explained by the scarcity of training examples only for

$d = 7, 14$ and 27. In contrast, the minimum distance $d = 2$ and 22 codes are well-represented in the dataset. These performance variations may indicate a different structure of the codes with these minimum distances from others.

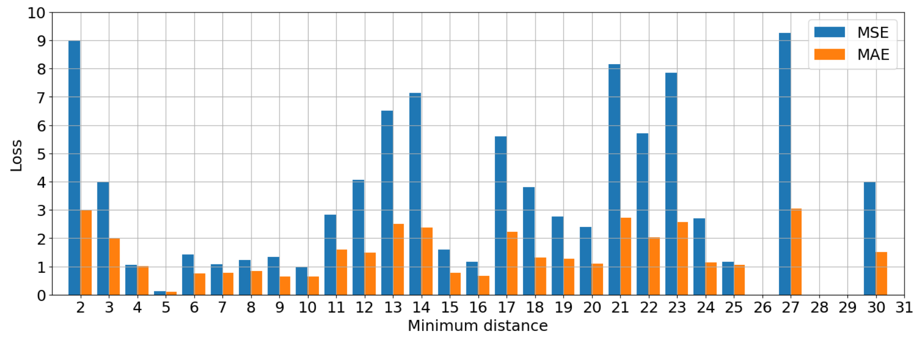

Figure 3: Losses on a test set for $\mathbb{F}_7$ codes with different minimum distances for the transformer model.

Figure 4 reveals a similar pattern for $\mathbb{F}_8$ case. The codes with minimum distances $d = 7, 14, 21$ and 28 codes are more difficult for the model to predict accurately. This behaviour cannot be attributed to data scarcity, which may again indicate that these codes are different structurally from those with other minimum distances.

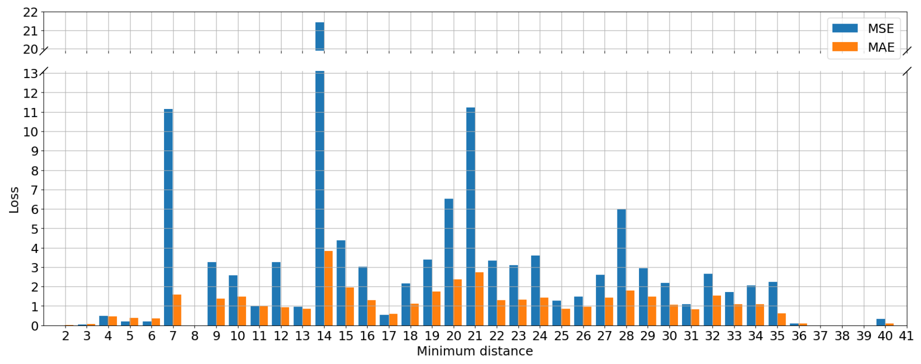

Figure 4: Losses on a testset for $\mathbb{F}_8$ codes with different minimum distances.

2.1.3 LSTM for Codes over $\mathbb{F}_7$

There are several architectures suitable for the sequence classification problem which we consider. We started our experiments with a smaller dataset, and used LSTM model to produce a baseline to compare with future experiments.

Initially generated dataset of 100,000 tuples (V, G_V, P_V, d_V) were generated by drawing a random number $m \in [3, 28]$ of vertices and then selecting random vertex coordinates. See figure 5.

The result is an imbalanced dataset with many minimum distances not being represented, which demonstrates the complexity of generating a representative dataset for all minimum distances. For instance, a rare minimum distance $d = 2$, which appears only once, originates from the code with dimension $k = 22$. There were 3755 codes with dimension $k = 22$, and the majority (88%) of them had a minimum distance $d = 6$. Codes with dimension $k = 23$ may seem more likely to produce a code with a minimum distance $d = 2$, but there were no such examples in the generated dataset.

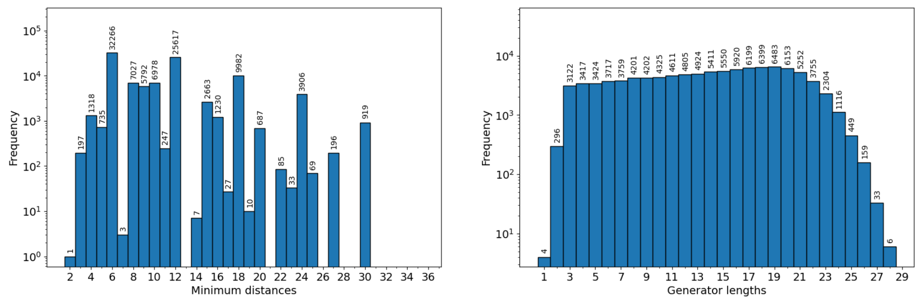

Figure 5: Distributions of Minimum distances d and dimensions of codes k (equal to the number of rows in generator matrix G) in the initial dataset of 100,000 codes over $\mathbb{F}_7$. Generated by drawing a random number $m \in [3, 28]$ of vertices and then choosing random vertex coordinates

This initial $100,000$ codes dataset was randomly split into training and test sets in a 9:1 ratio. Because the dataset was very unbalanced and no measures to rebalance it were taken, the test set was missing minimum distances of 2, 7, 17, and 36 compared to the whole dataset.

The model takes as input generator matrices [5] in batches. The target outputs d in the dataset were rescaled to be in region $[0, 1]$ using a simple formula $\tilde{d} = (d - \mu)/\sigma$, where $\mu = (30 + 3)/2 = 16$ and $\sigma = (30 - 3)/2 = 13.5$.

The model consisted of two layers of stacked LSTM. After the second layer, the hidden and cell states were extracted, concatenated, and passed to a linear layer projecting the output onto one continuous variable. The hyperparameters are presented in Table 1.

On a training set, the model's MAE was 2.72 and the accuracy within a tolerance ($d_V \pm 3$) was 93.0%. In the test set, the model's MAE was 3.02 and the accuracy within a tolerance ($d_V \pm 3$) was 91.2%. See figure 6 for per-minimum-distance metrics. The model performed better for the lower part of the minimum distance spectrum and worse for the upper part. This model serves as a proof of concept that it is possible to learn to predict the minimum distance of a code, and the model has a generalisation capability for unseen

[5] Without any encoding. To be discussed later.

Hyperparameter Type	Parameter Description	Value
Network architecture	Input dimension	36
	Hidden dimension	128
	Number of layers	2
	Batch size	4
Training parameters	Optimizer	AdamW
	Learning rate	$1 \cdot 10^{-5}$
	Epochs	200

Table 1: LSTM architecture and training hyperparameters for $\mathbb{F}_7$ codes minimum distance prediction.

data.

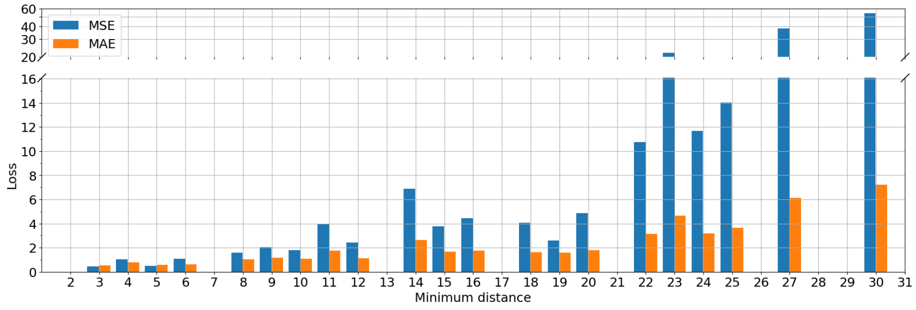

Figure 6: Losses on a test set for $\mathbb{F}_7$ codes with different minimum distances for the LSTM model.

Ultimately, the goal of the model is to be an estimator of the minimum distance, to be used in conjunction with the genetic algorithm (GA). During our experiments with GA, we observed that the final population after the GA run was somewhat noisy – that is, the dimensions of the resulting codes' minimum distances were distributed in some regions around the maximum value. The range between the smallest and the largest minimum distance typically varied from 3 to more over 10. As a result, the model does not need to be 100% accurate to be efficient in this context; a MAE error within a reasonable range would still be sufficient for practical use.

2.1.4 Toric Transformer

Recent advances in NLP are driven by the transformer architecture, which has demonstrated high effectiveness on various sequence processing tasks. It is therefore reasonable to test it in the current setting, which can be viewed as a sequence classification task.

Compared to NLP transformers, there are several architectural changes compared we implement for this task:

- input encoding of the field elements (for $\mathbb{F}_8$),
- masking to prevent padding from affecting results,
- summarization along time to produce a sequence length independent output,

which we discuss below in the description of the architecture[6].

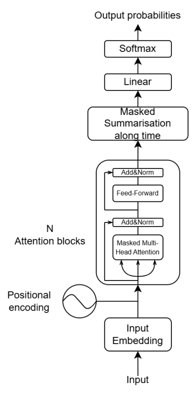

Figure 7: Toric Transformer - model architecture.

General structure of the model is depicted in Figure 7 and is shared by both models for $\mathbb{F}_7$ and $\mathbb{F}_8$ codes.

Input

The input of the model was composed of a code tensor and a length vector. The code tensor is a PyTorch tensor containing a batch of generator matrices, padded with zeros to standardize their dimensions. This tensor had tree dimensions denoted as (B,T,C), where

- B - batch dimension (i.e. the number of examples in a batch),
- T - time dimension, corresponding to the maximum number of rows in the generator matrices (i.e. $(q-1)^2$) [7],
- C - number of channels, corresponding to the length of each row in the generator matrix (i.e. $(q-1)^2$ as well).

The length vector stores the actual number of rows (i.e., the code dimension k) of each generator matrices in the batch. It can be used to distinguish where the true generator matrix rows and padding within the code tensor.

Input embedding

The input sequence of the rows of the generator matrix in a batch is passed through an embedding layer. Embedding is a useful technique originating from NLP. The idea is to map the elements of the input sequence into some higher-dimensional space. In NLP, text data is first split into elementary building blocks of the sequence - tokens, which are then represented as vectors in higher dimensional vector space, allowing the ML model to process them effectively. In our setting, the rows of the generator and parity-check matrices are already in a vector form with entries of the vectors being elements of $\mathbb{F}_q$ field. However,

[6]Short architecture description was given in the section **Model Architecture** of the main paper as well.

[7]For individual generator matrices, this dimension has a size equal to the code dimension k, but when padded and packed in the code tensor, it is equal to $(q-1)^2$.

- elements of $\mathbb{F}_7$ field are numbers in the range $[0, 6]$ and thus the rows can be used as an input *as is*;
- elements of $\mathbb{F}_8$ field cannot be written as integers because $\mathbb{F}_8$ field does not have a multiplicative generator. If we introduce α as a notation for the primitive element, then the elements of $\mathbb{F}_8$ field can be written as 0, 1, α, α^2, ..., α^6, $\alpha^7 = 1$. This representation was used in the Magma CAS. However, these elements are not linearly independent, and although one-hot encoding was found to be efficient in $\mathbb{F}_7$ case, using is not reasonable. The alternative we used was to include an embedding layer with learnable representation, which is less efficient[8] compared with a one-hot representation.

Attention blocks

The attention mechanism in the Transformer architecture and embedding align naturally with the structure of the problem of predicting minimum distance. By passing the rows of the generator matrix to the model, it will first represent them as some vectors via the learned representation layer. These representations can then be compared in attention heads between each other, and useful information for the prediction of the minimum distance can be extracted.

In our model, after the addition of positional encoding, the result was passed through an encoder-only set of self-attention blocks, each containing an attention layer and a feed-forward neural network.

Pooling layer

In standard transformer architecture, the number of dimensions of the input, which is of shape (B,T,C) is out case, is preserved on all the stages. But for classification tasks, a single fixed-size vector per sequence is needed, so we must eliminate the time dimension T in the output, i.e. introduce a pooling layer. There are many variants in the literature, but the simplest are:

- BERT-like pooling – introduce the special classification token [CLS] in the input, and then use the output at [CLS] for classification,
- average pooling – simply take the average of all output along T dimension and use for classification,
- attention pooling – applying an attention mechanism to compute a weighted sum of sequence elements.

However, it was shown [14], that the later two are somewhat better than the first approach, so we used them in our model.

[8] In a sense that training to the same level of loss generally takes more epochs for learnable embedding layer.

- We used average pooling in in $\mathbb{F}_7$ case and it demonstrated promising results despite being a very loose form of summarisation. The reason for this might be that within the attention layers, the rows of the generator matrix exchange information with each other and form more or less similar representations of the code;
- we expected the $\mathbb{F}_8$ case to be more complicated, so the attention pooling was implemented. The idea is essentially to do a transformation $v \to \text{softmax}(v^T M)v$ with a learnable square matrix M. It can reproduce the mean when M is proportional to the identity matrix and, therefore, should provide results that are not worse than the simple mean.

Output

After that, a linear classification layer, projecting from the embedding space to the output vector of dimension of $(q-1)^2$ (the number of possible minimum distances treated as $(q-1)^2$ independent classes), is added on top of the pooled output. Finally, a softmax function was applied to get probabilities of classes vector $\vec{p}$ which were then used to compute the expectation of the minimum distance as $\mathbb{E}[d] = \vec{p} \cdot \vec{i}$ where $\vec{i}$ is the class labels (target minimum distances) vector.

Since we were working with padded data, we had to implement masking to prevent padding elements from influencing the results in the attention blocks and pooling layers.

Loss functions

Since we are computing the expectation of the minimum distance, we could have used just mean square error loss between the expectation and the true minumun distance. However, this loss would discard all the information about the individual probabilities of the classes. To use this information, one can use Wasserstein loss (or distance) or a Cross-Entropy loss.

Wasserstein loss, sometimes referred as Earth Mover's Distance, has more applications with Generative Adversarial Networks. It measures the distance defined between probability distributions using some metric (we choose just Euclidean distance). On the other hand, Cross-Entropy loss measures the difference between two probability distributions, but it is not a symmetric metric. It is more suitable for classification problems and thus has more use in NLP.

The current problem naturally suggest using Wasserstein loss to bring the model's output distribution to point-mass distribution, which we done for $\mathbb{F}_7$ case. However, experiments demonstrated a superior performance of Cross-Entropy loss which we continued to use for $\mathbb{F}_8$ codes.

Hyperparameters

See Table 2 and 3 for the hyperparameters of transformer models for codes over $\mathbb{F}_7$ and $\mathbb{F}_8$, respectively.

Hyperparameter Type	**Parameter Description**	**Value**
Network architecture	Embedding	No
	Input dimension	36
	Embedding dimension	200
	Attention heads	10
	Transformer layers	2
	Dropout	0.2
	Loss function	Wasserstein-2
	Batch size	32
Training parameters	Optimizer	AdamW
	Learning Rate	$3 \cdot 10^{-4}$
	Epochs	22

Table 2: Transformer architecture and training hyperparameters for $\mathbb{F}_7$ codes minimum distance prediction.

Improvements

We used the models described above to search for champion codes using a genetic algorithm. We started experiments with $\mathbb{F}_7$ codes model, iteratively improving it, at both architecture and training procedure. These experiments showed that a number of modifications can be made to improve performance.

We found that instead of using the rows of the generator matrix as it is, it is more efficient to encode elements of rows using one-hot encoding. That is, we replaced entries in the input row by vectors of length seven containing zeros everywhere except one position unique to each of the numbers 0-6 (and flattened afterwards). As a result, the input vector of length 36 is replaced by a vector of length $36 \cdot 7 = 252$.

As described in **Toric Code Datasets** of the main paper, we used an oversampling technique to rebalance the dataset. To avoid biases in the model's output distribution caused by duplicated examples, we down-weighted the oversampled examples during training.

Using attention pooling instead of average pooling also demonstrated improved results.

Finally, in case of $\mathbb{F}_7$, we used the whole datasets divided into training and test sets. But unless we oversample some classes by a factor of thousands, the datasets remains extremely unbalanced. To overcome this problem, we introduced the two-stage training procedure:

- in the pre-training stage, we used classes which has many representatives (more than 10,000 for $\mathbb{F}_7$), and train model to learn general patterns useful for minimum distance prediction;
- in the training stage, we aimed to have all available classes represented, so we down-sampled classes with many representatives (to 500 for $\mathbb{F}_7$), and upsampled classes with not enough representatives (to 500 for $\mathbb{F}_7$),

Hyperparameter Type	**Parameter Description**	**Value**
Network architecture	Embedding	Learnable embedding
	Input dimension	49
	Embedding dimension	196
	Attention heads	7
	Transformer layers	2
	Dropout	0.2
	Loss function	Cross-entopy
	Batch size	128
Training parameters	Optimizer	AdamW
	Learning Rate pre-trainig	$3 \cdot 10^{-5}$
	Learning Rate post-trainig	$3 \cdot 10^{-6}$
	Epochs pre-train	120
	Epochs post-train	28

Table 3: Transformer architecture and training hyperparameters for $\mathbb{F}_8$ codes minimum distance prediction.

and trained the pre-trained model on the resulting dataset. The second training stage after the pre-training has relatively small dataset (14,000 for $\mathbb{F}_7$) so the model is prone to overfitting, so early stopping should be used.

Overall, this showed excellent results for $\mathbb{F}_7$ codes with the next iteration of the model (not used with GA), achieving Cross-Entropy loss of 0.61 on the pre-train dataset and 1.26 on the train set, accuracy within a tolerance ($d_V \pm 3$) of 84.8% on the train set and 90.3% on test set [9].

The application of this methodology for $\mathbb{F}_8$, however, demonstrated less inspiring results. Already at the pre-training stage with plenty of data (dataset was roughly the same in size as for $\mathbb{F}_7$ model with two-stage training), model was prone to overfitting after 120 epochs.

In general, there are two ways to deal with it: increase dataset size or decrease the number of parameters in the model. Knowing that the task of predicting minimum distance for $\mathbb{F}_8$ is more complicated than for $\mathbb{F}_7$, decreasing the model size is not an option. Therefore, one might hope to improve the results by increasing the dataset size, but data generation is very computationally expensive for $\mathbb{F}_8$ codes.

References

1. Hill, R. *A First Course In Coding Theory* (Clarendon Press, Oxford, 1988).
2. Grassl, M. *Bounds on the minimum distance of linear codes and quantum codes* Online, available at `https://www.codetables.de`.

[9] A counter-intuitive higher accuracy on the test than one the training set is caused by the fact that train set has oversampled classes.

3. Brown, G. & Kasprzyk, A. M. Seven new champion linear codes. *LMS J. Comput. Math.* **16,** 109–117. doi:10.1112/S1461157013000041 (2013).

4. Zimmermann, K.-H. *Integral hecke modules, integral generalized Reed–Muller codes, and linear codes* tech. rep. (Techn. Univ. Hamburg-Harburg, 1996).

5. Grassl, M. in *Discovering mathematics with Magma* 287–313 (Springer, Berlin, 2006). doi:10.1007/978-3-540-37634-7_13.

6. Hernando, F., Igual, F. D. & Quintana-Ortí, G. Algorithm 994: fast implementations of the Brouwer-Zimmermann algorithm for the computation of the minimum distance of a random linear code. *ACM Trans. Math. Software* **45,** Art. 23, 28. ISSN: 0098-3500. doi:10.1145/3302389 (2019).

7. Hansen, J. P. Toric varieties Hirzebruch surfaces and error-correcting codes. *Appl. Algebra Engrg. Comm. Comput.* **13,** 289–300. ISSN: 0938-1279. doi:10.1007/s00200-002-0106-0 (2002).

8. Hansen, J. P. in *Coding theory, cryptography and related areas (Guanajuato, 1998)* 132–142 (Springer, Berlin, 2000).

9. Kitaev, A. Y. *Quantum communication, computing, and measurement* in *Proceedings of the 3rd International Conference of Quantum Communication and Measurement, New York: Plenum* (1997).

10. Little, J. B. Remarks on generalized toric codes. *Finite Fields Appl.* **24,** 1–14. ISSN: 1071-5797. doi:10.1016/j.ffa.2013.05.004 (2013).

11. Ruano, D. On the structure of generalized toric codes. *J. Symbolic Comput.* **44,** 499–506. ISSN: 0747-7171. doi:10.1016/j.jsc.2007.07.018 (2009).

12. Soprunov, I. & Soprunova, J. Toric surface codes and Minkowski length of polygons. *SIAM J. Discrete Math.* **23,** 384–400. ISSN: 0895-4801. doi:10.1137/080716554 (2008).

13. Joyner, D. Toric codes over finite fields. *Appl. Algebra Engrg. Comm. Comput.* **15,** 63–79. ISSN: 0938-1279. doi:10.1007/s00200-004-0152-x (2004).

14. Tang, Y. & Yang, Y. Pooling and attention: What are effective designs for llm-based embedding models? *arXiv preprint arXiv:2409.02727* (2024).