MACHINE LEARNING DISCOVERS NEW CHAMPION CODES

YANG-HUI HE, ALEXANDER M. KASPRZYK, Q LE, AND DMITRII RIABCHENKO

Abstract. Linear error-correcting codes form the mathematical backbone of modern digital communication and storage systems, but identifying champion linear codes (linear codes achieving or exceeding the best known minimum Hamming distance) remains challenging. By training a transformer to predict the minimum Hamming distance of a class of linear codes and pairing it with a genetic algorithm over the search space, we develop a novel method for discovering champion codes. This model effectively reduces the search space of linear codes needed to achieve champion codes. Our results present the use of this method in the study and construction of error-correcting codes, applicable to codes such as generalised toric, Reed–Muller, Bose–Chaudhuri–Hocquenghem, algebrogeometric, and potentially quantum codes.

1. Introduction

Coding theory forms the foundation of all modern communication. Error-correcting codes – used to detect and correct errors in data transmitted over Wi-Fi and LAN networks [41], under transatlantic waters [47], and across deep space [21, 35] – are a key part of coding theory, and the search for more efficient error-correcting codes is of great importance. A crucial property when determining an error-correcting code's capabilities is its minimum Hamming distance. Finding new linear codes whose minimum Hamming distance equals or exceeds that of previously known codes (for fixed block length and dimension) is computationally challenging, and is an NP-hard problem [57]; these codes are called *champion codes*. We present a new method for discovering champion linear codes, employing machine learning and genetic algorithms, as summarised in Figure 1.

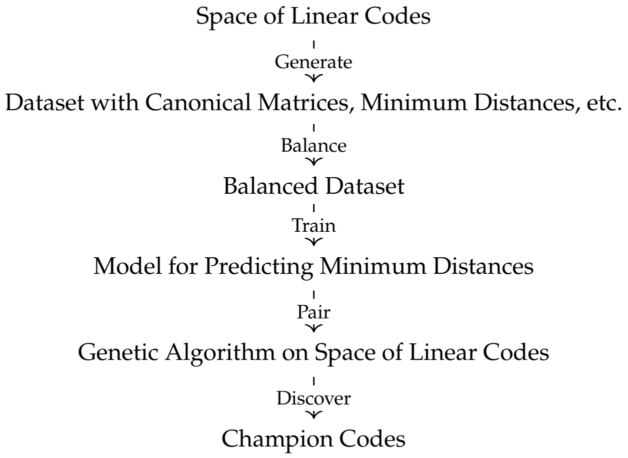
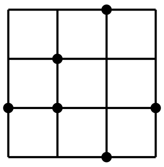

Space of Linear Codes
Generate
Dataset with Canonical Matrices, Minimum Distances, etc.
Balance
Balanced Dataset
Train
Model for Predicting Minimum Distances
Pair
Genetic Algorithm on Space of Linear Codes
Discover
Champion Codes

Figure 1. A general method for the discovery of champion error-correcting codes.

To demonstrate the efficacy of this method, we restrict ourselves to a class of error-correcting linear codes called generalised toric codes, introduced by Hansen [30] and Little [44], however there are many other classes of codes that could be used [4, 24, 50]. We develop a transformer architecture, adapted from a simplified version of OpenAI's GPT-2 model [49], to train a predictive model for the minimum Hamming distance d of a generalised toric code over $\mathbb{F}_7$, achieving ~91.6% accuracy with a tolerance of $d \pm 3$ and mean absolute error 1.05 on the test set. Combining this model with a genetic algorithm, we rediscover champion codes initially found in [12]. Using a similar method over $\mathbb{F}_8$ we find over 500 champion codes, of which at least six are new (see Table 6). A comparison of our method with a random search suggests that it achieves up to a twofold improvement in BZ evaluations, significantly reducing the computational cost of discovering new champion codes. A BZ

Key words and phrases. Machine learning; coding theory; toric codes; genetic algorithm; transformers.



$$\begin{array}{ccccc} V & & C_V(\mathbb{F}_5) & & d \\ & \longrightarrow & \begin{pmatrix} 1&0&0&0&0&1&3&0&0&2&0&4&4&2&2&1 \\ 0&1&0&0&0&0&1&3&0&2&1&3&2&1&1&0 \\ 0&0&1&0&0&4&3&2&0&4&0&2&0&2&1&1 \\ 0&0&0&1&0&1&1&2&0&1&1&4&1&0&2&1 \\ 0&0&0&0&1&2&4&3&0&0&0&0&4&3&1&2 \\ 0&0&0&0&0&0&0&0&1&2&4&3&1&2&4&3 \end{pmatrix} & \longrightarrow & 8 \end{array}$$

Figure 2. An example of a generalised toric code C_V over $\mathbb{F}_5$, with $V = \{(0,1),(1,1),(1,2),(2,0),(2,3),(3,1)\}$ and minimum Hamming distance eight.

evaluation is a use of the Brouwer–Zimmermann algorithm, the brute-force algorithm for determining the minimum Hamming distance of any linear code.

Our method builds upon previous work by Brown–Kasprzyk [12,13] which systematically classified all generalised toric codes over $\mathbb{F}_q$, $q \le 7$. The methods used there failed to extend to $\mathbb{F}_8$ due to the exponential computational cost of enumerating all candidate codes and calculating the minimum Hamming distance. Our method is, in principle, more general, and can be applied to any family of linear codes with an evolvable parameter space (see the section on 'Evolving champion codes' below).

This paper follows a tradition of applying the techniques of data science and machine learning to purely mathematical data. This tradition has its origins in the study of string theory and algebraic geometry [32], and has expanded to other areas of mathematics; for example [1,8,16,17,19,28,31,33,59, 60]. Though machine learning does not guarantee optimal solutions to all problems, our investigation shows a successful application of machine learning to an NP-hard problem [3] and a new methodology for discovering champion codes.

2. Methods

Generalised toric codes. We focus on generalised toric codes [44], a class of linear codes which have proven useful for discovering new champion codes [12, 13, 44, 52]. See §A of the Appendix for the basics of coding theory, as well as a summary of the standard notation.

Fix a finite field $\mathbb{F}_q$, for some prime power q, and primitive element $\xi \in \mathbb{F}_q^*$, and consider the planar lattice grid $[0, q-2]^2$. For each $u = (a,b) \in [0, q-2]^2$, define the map

$$\begin{aligned} e_u : \mathbb{F}_q^* \times \mathbb{F}_q^* &\longrightarrow \mathbb{F}_q \\ (\xi^i, \xi^j) &\longmapsto (\xi^i)^a (\xi^j)^b \end{aligned}$$

Let $V = \{(a_1,b_1),\ldots,(a_m,b_m)\}$ be a set of m lattice points in $[0,q-2]^2$. The *generalised toric code* $C_V(\mathbb{F}_q)$ over the field $\mathbb{F}_q$, associated to the lattice points V, is the linear code spanned by the vectors

$$\left\{(e_u((\xi^i,\xi^j)) : 0 \le i,j \le q-2) : u \in V\right\} = \left\{(\xi^{ia+jb} : 0 \le i,j \le q-2) : (a,b) \in V\right\}$$

This has block length $n = (q-1)^2$. An example of this construction is illustrated in Figure 2. When q is specified, we omit the field $\mathbb{F}_q$ from the notation.

If we restrict to (non-generalised) toric codes [29, 30], we can related them to divisors on toric varieties. In particular, results on cohomology and intersection theory for toric varieties can be used to obtain bounds on the minimum Hamming distance [30, 40, 54]. Although more is known in the context of (non-generalised) toric codes, we consider the generalised case for several reasons: constructing datasets for generalised toric codes is computationally cheaper; they have been successfully used to discover champion codes [12]; their simple parametrisation makes them suitable for genetic optimisation (see the section on 'Genetic algorithms' below); and they provide a non-trivial test case for our method.

We conclude this section by summarising the method we employ to find champion codes:

(1) consider a space of linear codes that have an evolvable parameter space, as explained in the section on 'Evolving champion codes' below;
(2) generate a dataset of each linear code's canonical generator matrix (or, equivalently, parity check matrix) and minimum Hamming distance;
(3) balance this dataset to allow for training and testing datasets;
(4) train a model to predict the minimum distance of a code with this balanced dataset;
(5) pair this model with a genetic algorithm that evolves over the space of linear codes;

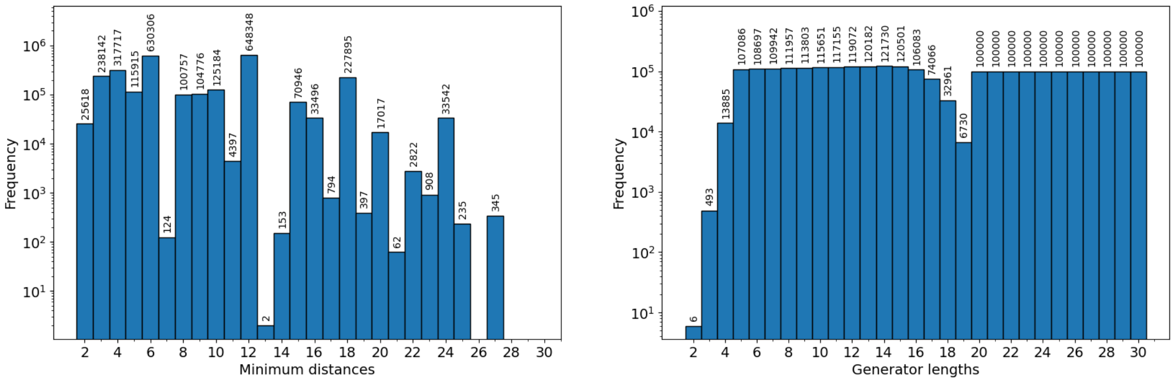


FIGURE 3. Histograms of the $\mathbb{F}_7$ codes dataset. The left histogram shows the distribution of the minimum Hamming distances. The right histogram shows the distribution of the generators' dimensions.

(6) run the genetic algorithm several times and discover new error-correcting codes, possibly finding champion codes.

Machine learning the minimum Hamming distance. The problem of computing the minimum Hamming distance of a code is an NP-hard problem and is typically calculated using the Brouwer–Zimmermann (BZ) algorithm [27, 61]. Despite several improvements, the BZ algorithm remains computationally costly, having polynomial in k and exponential in q complexity [36]. Thus our method seeks to minimise the number of BZ evaluations required to identify new champion codes (see Tables 7 and 8 for results).

A linear code C can be represented by a $k \times n$ generator matrix G (or, equivalently, a parity check matrix P) whose rows form a basis of C; here n is the block length of C and k is the dimension of C. Thus the problem can be reformulated as follows: given an input sequence S of k codewords, each of length n, predict the minimum Hamming distance d (a non-negative integer bounded above by n). This task belongs to a family of problems called *sequence modelling*, for which several architectures are commonly used: convolutional neural networks, recurrent neural networks, and transformers. The latter two naturally handle variable-length input sequences (although transformers impose a maximum sequence length).

Since computing a code's minimum Hamming distance is NP-hard, we employ the most powerful sequence modelling architecture currently available, the transformer [58], which underlies many state-of-the-art natural language processing (NLP) models such as ChatGPT [14] and sentiment analysis systems [5]. The latter task is conceptually analogous to ours: given a sequence of elements (text tokens or, analogously, codewords), the goal is to predict a class label (text sentiment or, analogously, minimum Hamming distance). Accordingly, we sometimes refer to codes of a given minimum distance as belonging to a *class*.

To train this model, we constructed a dataset consisting of triples (V, G, d), where V is a set of points in $[0, q-2]^2$ defining a generalised toric code $C_V(\mathbb{F}_q)$, G is the canonical generator matrix of C_V, and d is the minimum Hamming distance of C_V. See [34] for an implementation using `SageMath` [20], `Magma` [10], and `PyTorch` [48]. Further details of the models can be found in §B of the Appendix.

Datasets of generalised toric codes. The dataset generated for generalised toric codes over $\mathbb{F}_7$ contains 2,700,000 elements. This data is summarised in Figure 3. Notice that there is significant variation in the frequencies of the minimum Hamming distances, ranging from just two cases to more than 600,000. As the dataset is unbalanced, splitting the data into train and test sets randomly would be inadequate. Instead, we split subsets with equal minimum distance into train and test sets separately and merged them into full train and test sets with oversampling to 500 or downsampling to 50,000 if necessary, and re-shuffling afterwards. A resulting dataset of approximately 550,000 elements was used for training with 9:1 train/test split.

We generated a smaller dataset containing 1,584,099 generalised linear codes over $\mathbb{F}_8$; the dataset is summarised in Figure 4. The complexity for predicting codes over $\mathbb{F}_8$ is typically larger than over $\mathbb{F}_7$, and the amount of data we collected was marginally smaller than for $\mathbb{F}_7$. To mitigate this, we used a

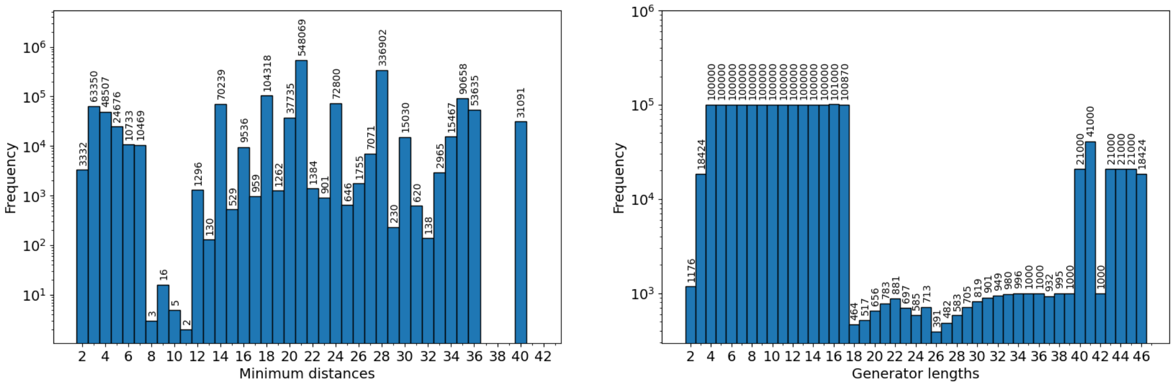


Figure 4. Histograms of the $\mathbb{F}_8$ codes dataset. The left histogram shows the distribution of the minimum Hamming distances. The right histogram shows the distribution of the generators' dimensions.

two-stage approach to training the model: first, to extract useful features of codes from the common examples in the pre-training stage; and second, to generalise this knowledge to all the classes in the training stage. Pre-training was done on a dataset with around 300,000 examples, consisting of classes with more than 10,000 examples. In the training stage, we used all the classes, down-sampling or over-sampling all available classes to 600, with a total dataset of 20,000 elements. We used 9:1 train/test split in both stages.

Model architecture. We formulated the prediction of the minimum Hamming distance as an NLP-style sequence classification task. The input sequence come from interpreting the canonical generator matrix G of a generalised toric code as a sequence S of length k whose elements (the rows of G) are codewords in $\mathbb{F}_q^{(q-1)^2}$, and the class label is the minimum Hamming distance $d \in \{0, \ldots, (q-1)^2\}$. The maximum value $d = (q-1)^2$ occurs only when $k = 1$; hence we restrict the classes to $\{0, \ldots, (q-1)^2 - 1\}$.

We modified a stripped-down GPT-2 [42, 49] to perform sequence classification. The adapted model uses: embeddings for field elements (none for $\mathbb{F}_7$, 4-dimensional vectors for $\mathbb{F}_8$); sinusoidal positional encodings of rows; two layers of attention blocks, each with masked self-attention and a feed-forward neural network; masked attention summarising information from each row; and a soft-max transformation to produce probabilities for minimum-distance classes. This is illustrated in Figure 5. See the Appendix for hyper-parameters used, the general structure of the network, and for a more detailed discussion.

3. Results

Training and performance. To measure performance, we report the proportion of estimates falling within a three-unit range ($d \pm 3$) of the actual minimum Hamming distance d (we call this the *accuracy*), alongside the mean absolute error (MAE) and mean squared error (MSE) per minimum-distance class.

Over $\mathbb{F}_7$, our model's train set achieved overall 1.04 MAE and 91.9% accuracy. Per class MAE and MSE can be seen in Figure 6. On the test set, consisting of around 61,000 examples, our model achieved 1.05 MAE and 91.6% accuracy. Notice that for all classes, the MAE is less than three and most MSEs roughly equal the square of the MAEs, indicating only a small number of larger errors are present in the model's predictions.

The $\mathbb{F}_8$ model's post-train stage achieved overall MAE 1.09 and 93.4% accuracy on the train set and MAE 1.21 and accuracy 92.4% on the test set. Figure 7 shows per class MAE and MSE measurements. Although these numbers trail the $\mathbb{F}_7$ results slightly, they remain competitive, and most minimum-distance classes stay well within the ±3-error target. Larger deviations are confined to a few classes ($d = 7, 14, 20, 21, 28$), showing opportunities for future improvement.

Evolving champion codes. We now describe how we coupled our predictive model with a genetic algorithm (GA) in order to search for optimal generalised toric codes by dimension. Repeated runs over $\mathbb{F}_7$ rediscovered champion codes seen in [12]; the same approach over $\mathbb{F}_8$ discovered new champion

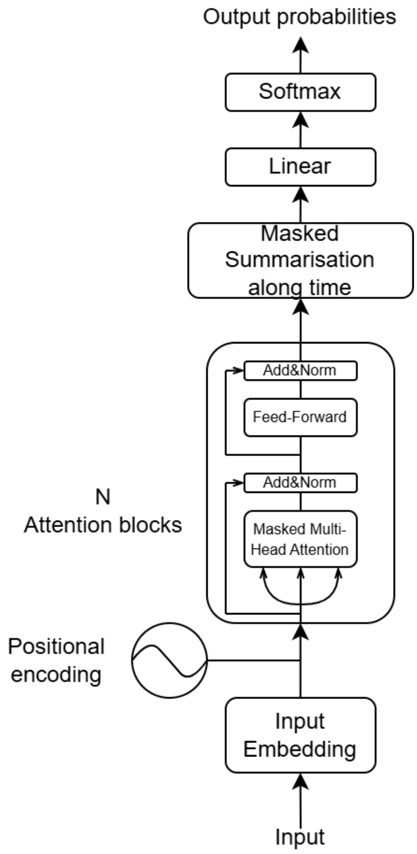


FIGURE 5. Toric transformer – model architecture.

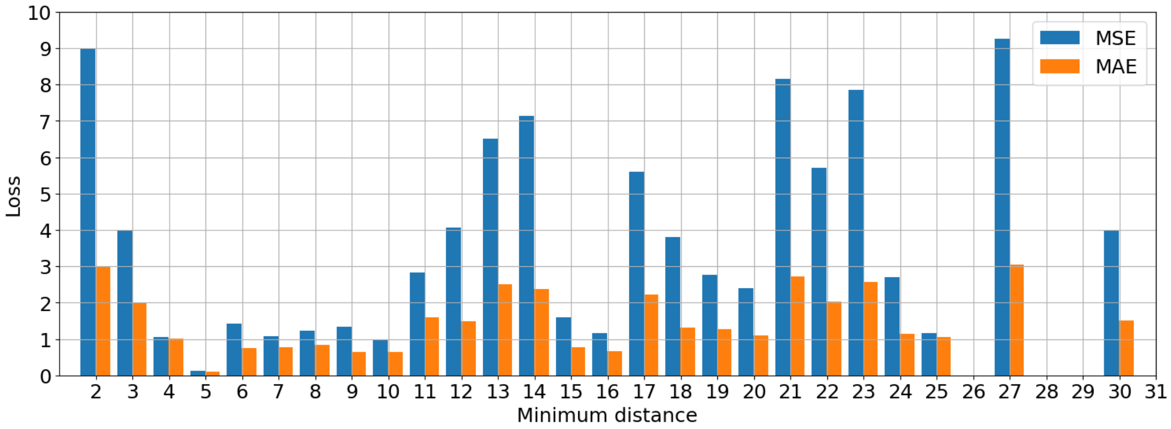


FIGURE 6. Losses on a test set for $\mathbb{F}_7$ codes.

linear codes. This approach may be adapted to other classes of codes, including Reed–Muller codes [46, 51], Bose–Chaudhuri–Hocquenghem codes [9,39], and algebrogeometric codes [18,25].

Genetic algorithms. Fix a positive integer m and a field $\mathbb{F}_q$, and consider a *population* $\{V_i\}$ consisting of sets V_i of m lattice points in $[0, q-2]^2$; in practise $m = k$ is our target dimension; –this is justified below. A set $V = \{(a_1, b_1), \dots, (a_m, b_m)\}$ is called a *chromosome*, and a lattice point $(a_j, b_j) \in [0, q-2]^2$ is called a *gene*. With the fitness function of our genetic algorithm as our model's predicted minimum distance (see Algorithm 1), each generation, guided by a single parameter test with the models, proceeds as follows.

(1) **Selection.** Stochastic universal sampling picks 200 parents from a population of 300.

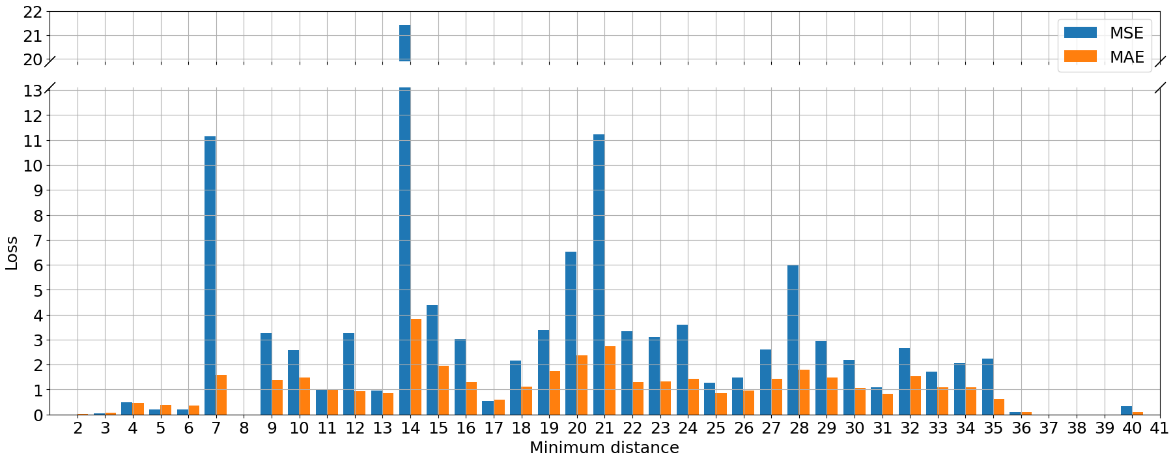


Figure 7. Losses on a test set for $\mathbb{F}_8$ codes.

Algorithm 1: Fitness function

```
Data: Target dimension $k$; a chromosome $V \subset [0, q-2]^2$ to be evaluated; a set of sets $S$ containing the
      chromosomes found by the GA so far.
Result: Updated value for $S$ and a fitness score $f$ for $V$.
if $V \in S$ then
    $f \leftarrow 10$
else
    $k_C \leftarrow \dim C_V(\mathbb{F}_q)$
    $G \leftarrow \mathsf{GeneratorMatrix}(C_V(\mathbb{F}_q))$        // calculate the canonical generator matrix
    $d_{\mathrm{approx}} \leftarrow \mathsf{Model}(G)$                      // estimate the Hamming distance
    $S \leftarrow S \cup \{V\}$
    $f \leftarrow 300 + d_{\mathrm{approx}} - |k_C - k|$
```

(2) **Crossover.** Flatten each parent's chromosome into a list $[a_1, b_1, \ldots, a_m, b_m]$ of length $2m$. A random crossover split point made between genes swaps the tails; offspring with duplicate lattice points $(a_i, b_i) = (a_j, b_j)$, $i \neq j$, are discarded and we repeat with other parents.

(3) **Mutation.** With 10% probability, a lattice point is replaced by a new, unused one, preserving m distinct lattice points.

(4) **Elitism.** The best 30 chromosomes carry forward unchanged, with the rest discarded.

We repeat for 200 generations. The genetic algorithms we used can be obtained from [34].

Note that we evolve lattice point sets V rather than generator matrices G: there is no clear way to mutate a generator matrix whilst preserving the class codes considered. By mutating V we can guarantee that we remain in the class of generalised toric codes. More generally, when there exists a way to mutate a code whilst remaining within a specified class, we say that that class has an *evolvable parameter space*. Spaces of this sort include: generalised toric codes; Reed–Muller codes; Bose–Chaudhuri–Hocquenghem codes; and algebrogeometric codes. It could be possible that certain classes of quantum codes also have an evolvable parameter space.

We do not store all codes generated during a run of the GA; only those produced for BZ evaluation will be stored in our datasets. We target champion codes of dimension k by setting $m = k$ and penalising candidates whose dimension drifts from k. To see why the dimension k should equal the number of lattice points m, observe that: on the one hand, increasing m does not improve our chances of producing a champion code of dimension k; on the other hand, we must have $k \leq m$. Thus, optimal codes should be found when $k = m$. Additional penalties prevent rediscovering codes found in earlier runs. This leads to the fitness function described in Algorithm 1, and used over both $\mathbb{F}_7$ and $\mathbb{F}_8$; note that the number 300 in the fitness score is simply a way to offset the other two terms and ensure that the output of the formula is positive, as required by the `PyGAD` library [23]. During GA search, we predict each code's minimum Hamming distance with our model (d_{approx}); promising candidates are then verified with the computationally expensive BZ algorithm. Since evaluating the minimum

Algorithm 2: Discovering best codes over $\mathbb{F}_7$ for dimensions $3 \le k \le 34$

```
Data: A list $(d_k)$ of the largest known minimum Hamming distance $d_k$ in dimension $k$.
Result: A list $(d'_k)$ of the largest minimum Hamming distance $d'_k$ discovered by the GA in dimension $k$; a
        list $(S_k)$ of discovered codes in dimension $k$ with minimum Hamming distance $d'_k$.

def UpdateDistances($S$, $k$):
    for $V \in S \setminus S^{\mathrm{seen}}$ do
        $d_C \leftarrow \mathrm{BZ}(C_V(\mathbb{F}_7))$                    // calculate the minimum Hamming distance
        if $d_C = d'_k$ then
            $S_k \leftarrow S_k \cup \{V\}$
        else if $d_C > d'_k$ then
            $d'_k, S_k \leftarrow d_C, \{V\}$
    $S^{\mathrm{seen}} \leftarrow S^{\mathrm{seen}} \cup S$

initialise the data
$S^{\mathrm{seen}} \leftarrow \{\}$
for $k \leftarrow 3$ to $34$ do
    $d'_k, S_k \leftarrow 0, \{\}$

iteratively update the distances
repeat
    for $k \leftarrow 3$ to $34$ do
        if $d'_k \neq d_k$ then
            $S \leftarrow \mathrm{GA}(k)$                          // generate codes of dimension $k$
            UpdateDistances($S$, $k$)
until some termination condition is satisfied
```

k	#Runs	#Codes
3	1	98
4	1	332
5	1	23446
6	1	4023
7	2	115
8	1	564
9	7	27
10	1	2482
11	3	124
12	55	0
13	1	411
14	2	16
15	82	1
16	1	11838
17	1	1951
18	20	66
19	n/a	n/a
20	1	1094
21	15	39
22	18	57
23	1	336
24	43	5
25	1	45714
26	1	21432
27	1	4215
28	6	80
29	1	2420
30	1	33302
31	1	10713
32	1	0
33	1	0
34	1	0

Table 1. The dimension k of the code, the number of iterations required to find the best minimum Hamming distance, and the number of codes found that obtain the minimum Hamming distance, over $\mathbb{F}_7$.

Hamming distance is the main bottleneck, and this is much costlier to check over $\mathbb{F}_8$, we apply the BZ check more selectively in that case.

Champion codes over $\mathbb{F}_7$. We applied our $\mathbb{F}_7$ model to rediscover the best generalised toric codes over $\mathbb{F}_7$ from [12]; see Algorithm 2.

Across 757 GA runs (see Table 1), most dimensions yielded a best-distance code on the first attempt. Dimensions 12 and 15 needed more iterations, indicating a tougher search landscape. Dimension 19 remains elusive after 501 runs, pointing to the rarity of $[36, 19, 12]$ generalised toric codes likely coupled with a tougher search landscape.

Table 1 shows a loose correlation: dimensions that contain a larger number of best codes in the dataset typically require fewer GA runs to rediscover them. This pattern fades at the smallest and largest dimensions, where optimal codes are naturally easier to locate than those in the 12–28 range. Thus, dimensions 12, 15, 19, and 28 converged more slowly because the dataset held few optimal-code

Algorithm 3: Discovering new champion codes over $\mathbb{F}_8$ for dimensions $3 \leq k \leq 46$

```
Data: A list $(d_k)$ of the best known lower bound on the minimum Hamming distance $d_k$ in dimension $k$.
Result: A list $(d'_k)$ of the largest minimum Hamming distance $d'_k \geq d_k$ discovered by the GA in
        dimension $k$; a list $(S_k)$ of discovered codes in dimension $k$ with minimum Hamming distance $d'_k$.

def UpdateCandidates($S$, $k$):
    for $V \in S \setminus S^{\text{seen}}$ do
        $C \leftarrow C_V(\mathbb{F}_8)$
        if VerifyMinimumDistanceLowerBound ($C, d_k$) then
            $S_k^{\text{candidate}} \leftarrow S_k \cup \{V\}$
    $S^{\text{seen}} \leftarrow S^{\text{seen}} \cup S$

initialise the data
$S^{\text{seen}} \leftarrow \{\}$
for $k \leftarrow 3$ to 46 do
    $S_k^{\text{candidate}} \leftarrow \{\}$

iteratively update the candidates
repeat
    for $k \leftarrow 3$ to 46 do
        $S \leftarrow$ GA($k$)                                   // generate codes of dimension $k$
        UpdateCandidates ($S, k$)
until some termination condiion is satisfied

verify the candidates
for $k \leftarrow 3$ to 46 do
    $d'_k, S_k \leftarrow d_k, \{\}$
    for $V \in S_k^{\text{candidate}}$ do
        $d_C \leftarrow$ BZ($C_V(\mathbb{F}_8)$)                 // calculate the minimum Hamming distance
        if $d_C = d'_k$ then
            $S_k \leftarrow S_k \cup \{V\}$
        else if $d_C > d'_k$ then
            $d'_k, S_k \leftarrow d_C, \{V\}$
```

examples at those sizes (this is amplified for dimensions 19 and 28 by the rarity of examples in the training set). Notably, though there were no best codes of dimension 12 in the training and testing datasets, our method still found a best generalised toric code of dimension 12. We see this more apparently over $\mathbb{F}_8$, where we discover new champion codes.

Champion codes over $\mathbb{F}_8$**.** Because champion codes over $\mathbb{F}_8$ are largely unknown, we screen candidates against the best-known lower bounds d_k from [26]. For a code C of dimension k, we use `Magma`'s `VerifyMinimumDistanceLowerBound`(C, d_k) intrinsic. If the BZ routine shows $d_C < d_k$, we discard C immediately; otherwise we keep it for a full distance check. This filter saves considerable computation by focusing effort only on codes that *could* beat the current records. See Algorithm 3.

We performed one run, due to time and computational constraints, of the above algorithm, excluding dimension 30 due to the high computational cost of running the BZ algorithm on this dimension. According to `Magma`'s time estimates, computing the 300 codes for dimension 30 would have exceeded our computational time budget. Hence there is no GA run for $l = 30$ in Table 8. Despite this, we found over 700 possibly new champion codes. As with $\mathbb{F}_7$, we compared these results with the number of champion codes found in the dataset on which the $\mathbb{F}_8$ model trained. Once again, champion codes were discovered at the lowest and highest dimensions for similar reasons as over $\mathbb{F}_7$. Rather surprisingly, in only one run, we achieved champion codes in dimensions that had no champion codes in their datasets: dimensions 18, 22, and 38. Through explicit calculation compared with the known best codes in [26], we see that all codes of dimension 18, 22, and 38 that have been discovered are in fact new: see Table 6.

Note that we had previously run the algorithm for $20 \leq m \leq 29$, where m is the size of the lattice point sets considered. In this search, we discovered a champion code of dimension 22, as seen in

k	3	4	6	7	8	18	22	38	40	41	44	46
Min. dist.	42	40	36	35	34	21	18	7	6	6	4	3
Singleton bound	47	46	44	43	42	32	28	12	10	9	6	4

Table 2. Code parameters with $n = 49$ and corresponding Singleton bounds.

q	7	8	9	11	13
Approx. size	$6.96 \cdot 10^5$	$2.49 \cdot 10^9$	$3.95 \cdot 10^{13}$	$7.94 \cdot 10^{23}$	$5.03 \cdot 10^{36}$

Table 3. Approximate size of $|X/\mathrm{AGL}_2(\mathbb{F}_q)|$

Table 5. Upon discovery, we saved this code, however due to an error on our part we were unable to save the corresponding dataset that included this code. As a result, this code is *not* noted in Table 8.

For any linear code, a standard upper bound on the minimum distance is given by the Singleton bound, $d \leq n - k + 1$. A comparison with our results (for $n = 49$) appears in Table 2. See [6,43,44,54] for sharper bounds based on additional code information.

4. Comparing our method with random search

Recall that our method aims to minimise the number of BZ evaluations required to identify new champion codes. To illustrate our method's advantage over random search, we first detail the size of the search space. Over $\mathbb{F}_q$, there are $2^{(q-1)^2}$ possible lattice point sets for constructing generalised toric codes. However, distinct sets $V_1 \neq V_2$ may yield equivalent linear codes. We approximate the number of inequivalent codes as

$$\left|\frac{X}{\mathrm{AGL}_2(\mathbb{F}_q)}\right| \approx \frac{2^{(q-1)^2}}{q^2(q^2-1)(q^2-q)},$$

where $\mathrm{AGL}_2(\mathbb{F}_q)$ is the affine general linear group of $\mathbb{F}_q$ in two dimensions. One may derive this formula as follows. By Burnside's Lemma,

$$\left|\frac{X}{\mathrm{AGL}_2(\mathbb{F}_q)}\right| = \frac{1}{|\mathrm{AGL}_2(\mathbb{F}_q)|} \sum_{g \in \mathrm{AGL}_2(\mathbb{F}_q)} |X^g|,$$

where $X^g = \{x \in X : g \cdot x = x\}$. Now,

$$\begin{aligned} |\mathrm{AGL}_2(\mathbb{F}_q)| &= |\mathbb{F}_q^2| \cdot |\mathrm{GL}_2(\mathbb{F}_q)| \\ &= q^2(q^2-1)(q^2-q). \end{aligned}$$

Consider $|X^g|$ for each $g \in \mathrm{AGL}_2(\mathbb{F}_q)$: if g is the identity, then all configurations are fixed, so $|X^g| = 2^{(q-1)^2}$; if g is not the identity, then we have significantly fewer fixed configurations, which we may approximate as negligible as q increases.

Approximate values for the number of unique generalised toric codes over $\mathbb{F}_q$ ($q = 7, 8, 9, 11, 13$) are shown in Table 3. Note that this estimate is less accurate for small q, where fixed configurations under affine transformations are non-negligible, but becomes effective for large q, where exact enumeration is infeasible. To our knowledge, no prior work has quantified the proportion of generalised toric codes with a given minimum Hamming distance among codes of fixed dimension or $|V|$.

One way to compare the efficiency of the GA over a random search is to estimate the expected number of BZ evaluations required to find a champion code. Let $N_k^{(m)}$ denote the number of distinct codes of dimension k generated from lattice point sets of size m, and let $C_k^{(m)}$ denote the number of champions amongst them. The expected number of evaluations in a random search is given by

$$\mathbb{E}[\text{steps before champion}] = \frac{C_k^{(m)}}{N^{(m)}}, \qquad \text{where } N^{(m)} = \sum_k N_k^{(m)}.$$

For the GA, this expectation can be estimated using the number of codes evaluated and champions found, $\widetilde{C}_k^{(m)}$, giving

$$\mathbb{E}[\text{steps before champion for GA}] = \frac{\widetilde{C}_k^{(m)}}{N^{(m)}}.$$

Tables 7 and 8 summarise these statistics for both random search and our GA search over $\mathbb{F}_7$ and $\mathbb{F}_8$.

For smaller and larger values of m, where champions are common, using the GA provides little advantage. However, these cases are not computationally difficult. In contrast, for intermediate values of m the GA achieves up to a twofold reduction in BZ evaluations relative to random search. This can be attributed to two factors:

(1) in certain dimensions (such as $k = 16$ over $\mathbb{F}_7$ and $k = 40$ over $\mathbb{F}_8$), the high density of champions causes the GA to saturate once it retains 30 elite codes per generation; and
(2) in sparsely represented dimensions (such as $16 \leq k \leq 39$ over $\mathbb{F}_8$), few or no champions appear in the training data, limiting the ML model's predictive accuracy, so that champion codes receive low approximations from the model.

An ideal case over $\mathbb{F}_7$ is $k = 14$, where the dataset contained few champion codes, making random discovery difficult. Nevertheless, the GA located a champion within two iterations. For $\mathbb{F}_8$, the mid-range $18 \leq m \leq 39$ is especially sparse of codes. Given the exponentially larger search space compared to $\mathbb{F}_7$, one might expect the discovery of champions to require substantially more BZ evaluations, making the discovery of champions in dimensions 18, 22, and 38 particularly striking. Despite limited representation in the dataset, the ML model successfully generalised from other dimensions to poorly represented ones, allowing our method to evolve new champion codes.

5. Discussion

Our method provides a new tool for the construction of champion error-correcting codes. We note some ways in which this approach can be improved: larger and more balanced datasets would give predictive models richer coverage, especially at rare minimum Hamming distances; broader hyper-parameter searches (optimisers, mixed cross-entropy and Wasserstein losses, alternative row/field encodings) and systematic embedding design could further improve accuracy; and improvements to the BZ algorithm [11] can accelerate the discovery of champion codes by reducing the computational cost of determining the minimum Hamming distance.

We wish to quantitatively characterise the distribution of generalised toric codes with respect to their minimum Hamming distances for fixed dimension k or fixed lattice point set size m – a question that has not yet been explored. The GA itself invites tuning: a wider sweep of population sizes, crossover strategies, and penalty weights, plus multi-week runs over $\mathbb{F}_8$ with staged `Magma` checks, should surface additional champions. It may also be fruitful to consider other ways to investigate the parameter space [53].

The same framework will apply essentially as-is to larger prime-powered finite fields $\mathbb{F}_q$, $q > 8$; it can be expanded to (non-generalised) toric codes, where we can exploit algebrogeometric properties arising from their toric description [30, 43, 54]; and it can be adapted to construct possible champion codes from other families with an evolvable parameter space (BCH, Reed–Muller, and more), once suitable GA 'genes' are defined. A different approach due to Vaessens et al. [56] gives a general GA scheme for building large codes, but this would be unsuitable for searching for codes of a fixed dimension as the fitness function would require a dimension penalty, as done in Algorithm 1. Finally, recasting the task as an image classification problem with padded matrices, or applying representation-learning techniques from NLP [7], offer alternative modelling avenues worth exploring.

Appendix A. Generalised toric codes

A *linear (error-correcting) code* over the finite field $\mathbb{F}_q$ is a k-dimensional linear subspace C of the vector space $\mathbb{F}_q^n$; here q is a prime power. The vectors in C are called *codewords* of C; the dimension n of the ambient vector space $\mathbb{F}_q^n$ is called the *block length* of C; and the number of possible codewords, $|C| = q^k$, is called the *size* of C. Given a codeword $w = (w_1, \ldots, w_n) \in C$, the *weight* of w is the number of non-zero entries w_i, that is, $|\{i : 1 \leq i \leq n, w_i \neq 0\}|$. The *minimum Hamming distance* between two codewords $w, w' \in C$ is equal to the weight of $w - w'$, that is, $|\{i : 1 \leq i \leq n, w_i \neq w'_i\}|$. The *minimum Hamming distance* d of the linear code is the minimum weight of its nonzero codewords or, equivalently, the minimum Hamming distance between any two distinct codewords. The block

k	#Champions
1	0
2	0
3	16464
4	31091
5	0
6	1034
7	216
8	28
9	3
10	11
11	1
12	144
13	1
14	0
15	146
16	1
17	0
⋮	⋮
39	0
40	6138
41	357
42	0
43	0
44	5177
45	0
46	16464
47	0
48	1176

Table 4. The dimension k and number of champion codes in training and testing datasets, over $\mathbb{F}_8$.

k	d	N	Lattice points V
3	42	231	(1, 0), (2, 3), (6, 3)
4	40	66	(0, 5), (2, 6), (6, 0), (6, 4)
6	36	3	(0, 0), (0, 6), (2, 4), (2, 5), (4, 6), (6, 4)
7	35	1	(0, 3), (2, 5), (4, 6), (5, 0), (5, 1), (6, 0), (6, 3)
8	34	1	(0, 4), (0, 6), (2, 3), (3, 4), (4, 0), (5, 1), (5, 3), (6, 0)
18	21	4	(0, 1), (0, 2), (0, 3), (0, 5), (1, 0), (1, 1), (1, 5), (1, 6), (3, 2), (3, 3), (3, 5), (3, 6), (4, 3), (4, 4), (5, 1), (5, 3), (5, 5), (6, 4)
22	18	1	(0, 0), (0, 6), (1, 0), (1, 3), (1, 4), (1, 5), (2, 0), (2, 5), (2, 6), (3, 2), (3, 3), (3, 5), (3, 6), (4, 2), (5, 0), (5, 3), (5, 4), (6, 0), (6, 1), (6, 3), (6, 4), (6, 5)
38	7	1	(0, 1), (0, 3), (0, 4), (0, 6), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 0), (2, 1), (2, 3), (2, 4), (2, 6), (3, 0), (3, 1), (3, 3), (3, 4), (3, 5), (3, 6), (4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (4, 6), (5, 0), (5, 1), (5, 3), (5, 4), (5, 5), (5, 6), (6, 0), (6, 1), (6, 2), (6, 5)
40	6	94	(0, 0), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (1, 1), (1, 2), (1, 5), (1, 6), (2, 0), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6), (3, 0), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6), (4, 1), (4, 2), (4, 4), (4, 5), (4, 6), (5, 1), (5, 2), (5, 3), (5, 4), (5, 5), (5, 6), (6, 0), (6, 1), (6, 4), (6, 5), (6, 6)
41	6	3	(0, 0), (0, 1), (0, 2), (0, 4), (0, 5), (0, 6), (1, 0), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 0), (2, 1), (2, 2), (2, 4), (2, 6), (3, 0), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6), (4, 0), (4, 1), (4, 4), (4, 6), (5, 0), (5, 1), (5, 2), (5, 3), (5, 5), (5, 6), (6, 0), (6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (6, 6)
44	4	100	(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (1, 0), (1, 1), (1, 3), (1, 4), (1, 5), (1, 6), (2, 0), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6), (3, 0), (3, 1), (3, 4), (3, 5), (3, 6), (4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (4, 6), (5, 0), (5, 1), (5, 2), (5, 3), (5, 4), (5, 5), (6, 0), (6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (6, 6)
46	3	235	(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (0, 5), (0, 6), (1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (2, 6), (3, 0), (3, 1), (3, 2), (3, 3), (3, 4), (3, 6), (4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (4, 6), (5, 1), (5, 2), (5, 3), (5, 4), (5, 5), (5, 6), (6, 0), (6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (6, 6)

Table 5. Champion generalised toric codes found over $\mathbb{F}_8$ with block length $n = 49$, dimension k, and minimum Hamming distance d. In each case, the number of such champion codes discovered is recorded by N, and a representative set of lattice points is given by V.

length n, dimension k, and minimum Hamming distance d of C are denoted by the triple $[n, k, d]_q$, or simply by $[n, k, d]$ when the value of q is clear. For an introduction to coding theory, see [37].

Let C be a linear code with parameters $[n, k, d]$. A choice of basis of C gives rise to a $k \times n$ *generator matrix* G, which can be expressed in canonical form $G = [I_k|H]$, 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

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

k	d	Lattice points V
18	21	(0, 1), (0, 2), (0, 3), (0, 5), (1, 0), (1, 1), (1, 5), (1, 6), (3, 2), (3, 3), (3, 5), (3, 6), (4, 3), (4, 4), (5, 1), (5, 3), (5, 5), (6, 4)
18	21	(0, 1), (0, 5), (1, 3), (1, 4), (1, 6), (2, 1), (2, 5), (2, 6), (2, 7), (3, 0), (3, 1), (3, 4), (3, 6), (4, 5), (5, 1), (6, 0), (6, 1), (6, 5)
18	21	(0, 0), (0, 1), (0, 4), (1, 1), (1, 2), (2, 6), (3, 4), (3, 6), (4, 1), (4, 2), (4, 5), (4, 6), (5, 2), (5, 5), (6, 0), (6, 4), (6, 5), (7, 6)
18	21	(0, 1), (0, 2), (1, 3), (1, 4), (1, 6), (2, 4), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6), (4, 1), (4, 2), (4, 6), (5, 1), (5, 3), (5, 5), (6, 4)
22	18	(0, 0), (0, 6), (1, 0), (1, 3), (1, 4), (1, 5), (2, 0), (2, 5), (2, 6), (3, 2), (3, 3), (3, 5), (3, 6), (4, 2), (5, 0), (5, 3), (5, 4), (6, 0), (6, 1), (6, 3), (6, 4), (6, 5)
38	7	(0, 1), (0, 3), (0, 4), (0, 6), (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 0), (2, 1), (2, 3), (2, 4), (2, 6), (3, 0), (3, 1), (3, 3), (3, 4), (3, 5), (3, 6), (4, 0), (4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (4, 6), (5, 0), (5, 1), (5, 3), (5, 4), (5, 5), (5, 6), (6, 0), (6, 1), (6, 2), (6, 5)

Table 6. All champion generalised toric codes found over $\mathbb{F}_8$ with block length $n = 49$, dimension $k = 18, 22$, or 38, and minimum Hamming distance d. In each case, the corresponding set of lattice points is given by V.

	Random				GA			
m	#Codes	#Codes with $k = m$	#Champions	#BZ per champion	#Runs	#Codes	#Champions	#BZ per champion
5	107086	86867	19307	4.50	1	300	72	4.17
6	108697	81094	3126	25.94	1	300	21	14.29
7	109942	74451	82	907.94	2	600	1	600.0
8	111957	67578	368	183.64	1	300	2	150.0
9	113803	60257	20	3012.85	7	2100	1	2100.0
10	115651	52704	1210	43.56	1	300	14	21.43
11	117155	45315	60	755.25	3	900	1	900.0
12	119072	38301	0	n/a	55	16500	1	16500.0
13	120182	31447	125	251.58	1	300	1	300.0
14	121730	25748	2	12874.0	2	600	1	600.0
15	120501	20857	1	20857.0	82	24600	1	24600.0
16	106083	16310	1859	8.77	1	300	32	9.38
17	74066	12458	350	35.59	1	300	7	42.86
18	32961	9026	22	410.27	4	1200	1	1200.0
19	6730	6730	0	n/a	84	25200	0	n/a
20	100000	100000	1094	91.41	1	300	4	75.0
21	100000	100000	39	2564.10	15	4500	1	4500.0
22	100000	100000	57	1754.39	18	5400	1	5400.0
23	100000	100000	336	297.62	1	300	1	300.0
24	100000	100000	5	20000.0	43	12900	1	12900.0
25	100000	100000	45714	2.19	1	300	134	2.24
26	100000	100000	21432	4.67	1	300	59	5.08
27	100000	100000	4215	23.72	1	300	10	30.0
28	100000	100000	80	1250.0	6	1800	1	1800.0
29	100000	100000	2420	41.32	1	300	6	50.0
30	100000	100000	33302	3.00	1	300	110	2.73
31	100000	100000	10713	9.33	1	300	33	9.09

Table 7. Search efficiency over $\mathbb{F}_7$ for generalised toric codes, comparing a random search with a GA search. For each lattice point set size m, the table records the total number of generated codes, those with dimension $k = m$ (this is only relevant for the randomly generated codes), the number of champion codes discovered, and the average number of BZ evaluations per champion.

The *parity check* matrix P of G is an $(n - k) \times n$ matrix satisfying $GP^T = 0$. Since the rows of P generate $C^{\perp}$, P is typically considered a dual generator matrix. Given the canonical form of the

	Random				**GA**			
			#BZ per	**Min/max**			**#BZ per**	**Min/max**
k	**#Codes**	**#Champions**	**champion**	**distance**	**#Codes**	**#Champions**	**champion**	**distance**
5	100000	0	n/a	36/38	300	0	n/a	35/38
6	100000	1034	96.71	36/36	300	3	100.0	36/36
7	100000	216	462.96	35/35	300	1	300.0	35/35
8	100000	28	3571.43	34/34	300	1	300.0	34/34
9	100000	3	33333.33	31/31	300	0	n/a	16/31
10	100000	11	9090.91	30/30	300	0	n/a	15/30
11	100000	1	100000.00	29/29	300	0	n/a	13/29
12	100000	144	694.44	28/28	300	0	n/a	12/28
13	100000	1	100000.00	27/27	300	0	n/a	11/27
14	100000	0	n/a	24/26	300	0	n/a	12/26
15	100000	146	684.93	24/24	300	0	n/a	16/24
16	101000	1	101000.0	24/24	300	0	n/a	10/24
17	100870	0	n/a	22/23	300	0	n/a	8/23
18	464	0	n/a	17/21	300	4	75.0	21/21
19	517	0	n/a	15/21	300	0	n/a	8/21
20	656	0	n/a	15/20	300	0	n/a	9/20
21	783	0	n/a	15/19	300	0	n/a	8/19
22	881	0	n/a	14/18	300	0	n/a	11/18
23	697	0	n/a	14/17	300	0	n/a	9/17
24	585	0	n/a	13/16	300	0	n/a	10/16
25	713	0	n/a	12/16	300	0	n/a	7/16
26	391	0	n/a	10/15	300	0	n/a	5/15
27	482	0	n/a	8/14	300	0	n/a	4/14
28	583	0	n/a	7/14	300	0	n/a	4/14
29	705	0	n/a	7/13	300	0	n/a	4/13
30	819	0	n/a	7/12	n/a	n/a	n/a	0/12
31	901	0	n/a	7/12	300	0	n/a	4/12
32	949	0	n/a	7/11	300	0	n/a	4/11
33	980	0	n/a	7/11	300	0	n/a	3/11
34	996	0	n/a	7/10	300	0	n/a	4/10
35	1000	0	n/a	7/9	300	0	n/a	4/9
36	1000	0	n/a	7/8	300	0	n/a	5/8
37	932	0	n/a	6/8	300	0	n/a	4/8
38	995	0	n/a	6/7	300	1	300.0	7/7
39	1000	0	n/a	6/7	300	0	n/a	3/7
40	21000	6138	3.42	6/6	300	95	3.16	6/6
41	41000	357	114.85	6/6	300	3	100.0	6/6
42	1000	0	n/a	5/6	300	0	n/a	3/6
43	21000	0	n/a	4/5	300	0	n/a	2/5
44	21000	5177	4.06	4/4	300	101	2.97	4/4
45	21000	0	n/a	3/4	300	0	n/a	2/4
46	18424	16464	1.12	3/3	300	236	1.27	3/3

Table 8. Search efficiency over $\mathbb{F}_8$ for generalised toric codes, comparing a random search with a GA search. For each dimension k, the table records the total number of generated codes, the number of champion codes, and the average BZ evaluations per champion discovered. Note that we excluded $k = 30$ from the GA run due to the high computational cost of BZ evaluations. Note also that a champion code of dimension 22, found by the GA and given in Table 5, is omitted here, as noted in the section 'Champion codes over $\mathbb{F}_8$'.

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, and is standard to assume the canonical form when discussing the generator and parity check matrices.

The minimum Hamming distance d of a code C is a measure of the accuracy of the code for error correction. An important task is to find codes with d as large as possible for a given block length n and dimension k. Theoretical upper and lower bounds on d are known, and Grassl [26] maintains a database of the current best known d, alongside a description of the corresponding code. We refer to any code with a larger minimum Hamming distance than the current best known example as a *champion* code, following the terminology of [12].

To calculate d, we use the Brouwer–Zimmermann (BZ) algorithm [27,61]. This 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 amongst the enumerated codewords. The computation completes once the lower and upper bounds converge. Despite several significant improvement [27], the BZ algorithm remains computationally costly. It has polynomial in k and exponential in q complexity [36].

Hansen [29,30] introduced a special class of linear codes inspired by toric geometry. Given a lattice point $u = (a, b) \in [0, q-2]^2$, we associate a vector $w_u \in \mathbb{F}_q^{(q-1)^2}$ whose entries are given by evaluating the monomial $x^a y^b$ at each point of the torus $(\mathbb{F}_q^*)^2$. More precisely, fix a primitive element $\xi \in \mathbb{F}_q^*$ and define the evaluation map

$$\begin{aligned} e_u : \mathbb{F}_q^* \times \mathbb{F}_q^* &\longrightarrow \mathbb{F}_q \\ (\xi^i, \xi^j) &\longmapsto (\xi^i)^a (\xi^j)^b \end{aligned}$$

Then $w_u = (e_u((\xi^i, \xi^j)) : 0 \leq i, j \leq q-2)$. Now let $P \subset \mathbb{Z}^2 \otimes \mathbb{Q}$ be a convex lattice polygon, and suppose that P is contained in the $[0, q-2]^2$ square. The *toric code* $C_P(\mathbb{F}_q)$ *associated with* P is defined to be the subspace spanned by the vectors $\{w_u : u \in P \cap \mathbb{Z}^2\}$. This has block length $n = (q-1)^2$ and dimension k at most $|P \cap \mathbb{Z}^2|$.

Generalised toric codes, introduced by Little [44], are constructed in a similar way, but allowing for the possibility of omitting lattice points from P. In principle this allows for the deletion of short vectors in C_P, thus potentially increasing the minimum Hamming distance of the resulting code. Equivalently, let $V \subset [0, q-2]^2$ be a set of lattice points. The *generalised toric code* $C_V(\mathbb{F}_q)$ associated with V is defined to be the subspace spanned by the vectors $\{w_u : u \in V\}$. Clearly when $V = P \cap \mathbb{Z}^2$, where $P = \mathrm{conv}(V)$ is the convex hull of V, we have $C_P = C_V$ and the concepts of toric codes and generalised toric codes coincide.

Example. We work over $\mathbb{F}_3$ with primitive element $\xi = 2$ in $\mathbb{F}_3^*$, and consider the region $[0, 1]^2$ consisting of four lattice points. Then

$$e_{(a,b)}((\xi^i, \xi^j)) = (\xi^i)^a (\xi^j)^b = \xi^{ai+bj}, \qquad \text{where } (a, b) \in [0, 1]^2 \text{ and } (\xi^i, \xi^j) \in (\mathbb{F}_3^*)^2.$$

In particular, a lattice point $u = (a, b)$ gives rise to the vector $w_u = (1, \xi^a, \xi^b, \xi^{a+b})$ (here we have implicitly fixed an ordering of the elements in $(\mathbb{F}_3^*)^2$). Set $V = \{(0, 0), (0, 1), (1, 0)\}$. The generalised toric code C_V is the subspace spanned by the vectors $(1, 1, 1, 1)$, $(1, 1, 2, 2)$, and $(1, 2, 1, 2)$ in $\mathbb{F}_3^4$. Generator and parity check matrices (written in canonical form) for C_V are:

$$G = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \qquad \text{and} \qquad P = \begin{pmatrix} 1 & 2 & 2 & 1 \end{pmatrix}$$

The minimum Hamming distance is readily seen to be two. Hence, C_V is a $[4, 3, 2]_3$-code.

Searches within the class of generalised toric codes have proven successful at finding champion codes [2,12,15,44,52]. In particular, all generalised toric codes over $\mathbb{F}_q$, $q \leq 7$, and a subset over $\mathbb{F}_8$, were considered in [12], identifying seven new champion linear codes. The method used there is exhaustive, and is computationally expensive due to the vast number of possible generalised toric codes, making a systematic search prohibitive when $q \geq 8$. As one would expect, the calculation of the minimum Hamming distance d is a significant bottleneck in any search for new champion codes; for example, iIt is remarked in [12] that "often these calculations would take millions of years if the code achieved its apparent minimum distance".

Algorithm 4: Generating datasets of size N over $\mathbb{F}_q$, for random lattice point sets of size m.

```
Data: A positive integer $N$; a prime-power $q$; a positive integer $m \leq (q-1)^2$.
Result: A set $S^*$ of $N$ distinct triples of the form $(V, G, d)$.

generate the set of sets of random lattice points
$S \leftarrow \{\}$
while $|S| < N$ do
    $S \leftarrow S \cup \{\mathsf{RandomPoints}(m)\}$

compute the required data
$S^* \leftarrow \{\}$
for $V \in S$ do
    $C \leftarrow C_V(\mathbb{F}_q)$
    $G \leftarrow \mathsf{GeneratorMatrix}(C)$        // calculate the canonical generator matrix
    $d \leftarrow \mathsf{BZ}(C)$                     // calculate the minimum Hamming distance
    $S^* \leftarrow S^* \cup \{(V, G, d)\}$
```

Appendix B. Machine learning experiments for generalised toric codes

A dataset of generalised toric codes over $\mathbb{F}_7$. For each $5 \leq m \leq 31$ we generated a dataset of $N = 100,000$ distinct triples of the form (V, G, d), where V is a set of lattice points in $[0, 5]^2$, $|V| = m$, chosen uniformly at random, G is a generator matrix (in canonical form) for the corresponding code C_V, and d is the minimum Hamming distance of C_V; see Algorithm 4. We also experimented with including the parity check matrix P of C_V as a feature but, as expected given the relationship between G and P, this was redundant. Note that the number of subsets of lattice points of size m is $\binom{(q-1)^2}{m} = \binom{36}{m}$. Thus, only in the range $5 \leq m \leq 31$ do there exist at least 100,000 distinct sets of lattice points of size m.

The dataset was generated using `Magma` [10], which can perform BZ computations in parallel, on a computer cluster node with 96 Intel Xeon(R) Gold 6442Y processors (48 cores, 96 threads, 2.6-4.0 GHz). The calculation took approximately 24 hours. The resulting dataset, containing 2,700,000 examples, is illustrated in Figure 3. This dataset contains considerable variation in the frequency of the minimum Hamming distances, with the number of occurrences ranging from as few as two to over 600,000 instances.

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 Hamming distance into train and test sets separately, merging them into full train and test sets with re-shuffling afterwards. Class imbalance can be further mitigated through the application of oversampling/downsampling of the minority/majority classes. Model training for codes over $\mathbb{F}_7$ used cross-entropy loss, which allowed us to down-weight the losses for oversampled classes, and up-weight the losses for downsampled classes. The final dataset for $\mathbb{F}_7$ codes contained ~550,000 elements, 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. The intention was to obtain moderately imbalanced dataset with the smallest classes being no smaller than 1% of the largest class. The only class with less than 10 elements, corresponding to a minimum distance 13, was reserved for the test set only.

A dataset of generalised toric codes over $\mathbb{F}_8$. In the case of codes over $\mathbb{F}_8$, we were able to generate $N = 100,000$ codes only for $4 \leq m \leq 17$, following Algorithm 4. Due to the computation cost of the BZ algorithm in higher dimensions, for $17 \leq m \leq 39$ we reduced our target sample size to $N = 1,000$, with an additional limit on the BZ computation time of 1 minute. An additional 20,000 codes were generated for $m \in \{2, 3, 40, \ldots, 47\}$ afterwards.

The dataset was generated using `Magma` on a computer cluster node with 96 Intel Xeon(R) Gold 6442Y processors (48 cores, 96 threads, 2.6-4.0 GHz). The calculation took approximately 48 hours. The resulting dataset, containing 1,584,099 elements, is illustrated in Figure 4. The outlier at $m = 42$ is the result of a mistake during data collection, when 20,000 data points for $m = 41$ were added twice, and 20,000 points for $m = 42$ were erroneously discarded; this error was spotted only after the experiments were completed, and it was judged too computation expensive to repeat the processes, with no clear benefit in terms of the conclusions we could draw.. When $m = 2$ and 3 the sample size is smaller than 20,000 elements; a similar situation occurs when $m = 47$. The variability in the number of examples by

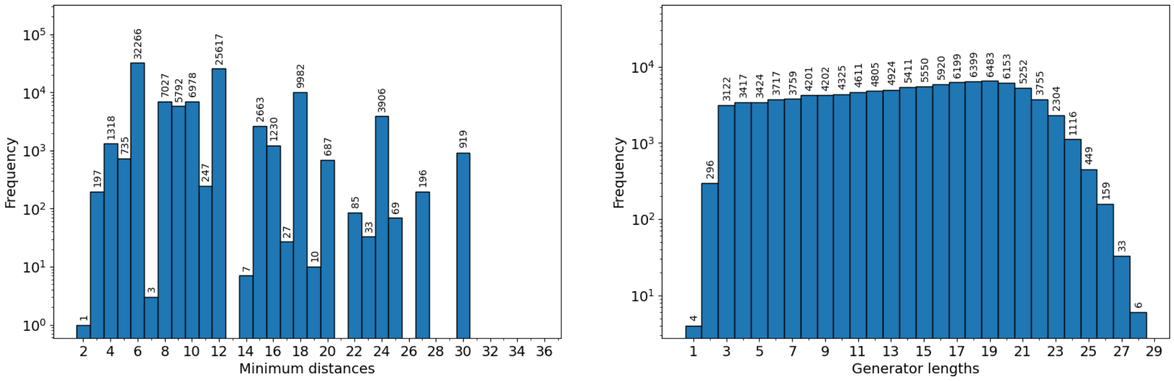


Figure 8. Distributions of the minimum Hamming distance d, and the dimensions k, of generalised toric codes in the initial randomly generated dataset of 100,000 codes over $\mathbb{F}_7$, as discussed in the section 'LSTM for codes over $\mathbb{F}_7$'.

minimum Hamming distance in the region $17 \leq m \leq 40$ is caused by the limit on the allowed runtime for the BZ algorithm, demonstrating the complexity of the procedure in this region. An unexpected feature of the $\mathbb{F}_8$ dataset is that dimension k of each code is always equal to the number of lattice points m used to generate the code – this stands in contrast to the $\mathbb{F}_7$ dataset.

The minimum Hamming distance prediction task for the codes over $\mathbb{F}_8$ is expected to be more complicated than for codes over $\mathbb{F}_7$, but the amount of data we managed to collect was marginally smaller than for $\mathbb{F}_7$. To compensate for this limitation, we used 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 Hamming distance) with more than 10,000 examples; subsets with more than 20,000 examples were downsampled to 20,000. For 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 generalise 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 in the section 'Training and performance' (see §3). Here we will point out a couple of features found in the results. Over $\mathbb{F}_7$, Figure 6 reveals a notable trend: the model exhibits inferior performance for codes with minimum Hamming 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, minimum Hamming distances $d = 2$ and 22 are well-represented in the dataset. These performance variations may indicate some intrinsic differences for codes with these minimum Hamming distances. Figure 7 reveals a similar pattern for codes over $\mathbb{F}_8$: the codes with minimum Hamming distances $d = 7, 14, 21$, and 28 are more difficult for the model to accurately predict. Again, this behaviour cannot be attributed to data scarcity.

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 a long short-term memory (LSTM) model [38] to produce a baseline against which to compare future experiments.

A dataset of 100,000 triples (V, G, d) was generated by repeatedly choosing a random value for $m, 3 \leq m \leq 28$, and then selecting m random distinct lattice points for V. This dataset is illustrated in Figure 8. The result is an imbalanced dataset with many minimum Hamming distances not represented; this demonstrates the complexity of generating a representative dataset of generalised toric codes. For example, a rare minimum Hamming distance $d = 2$, which appears only once, originates from a generalised toric code with dimension $k = 22$. There were 3755 codes with dimension $k = 22$, and the majority (88%) had minimum Hamming distance $d = 6$. Codes with dimension $k = 23$ may seem more likely to have minimum Hamming distance $d = 2$, but there were no such examples in the dataset.

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 Hamming distances of 2, 7, 17, and 36. For input, the model takes batches of

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

TABLE 9. The LSTM architecture and training hyperparameters used for predicting minimum Hamming distances for generalised toric codes over $\mathbb{F}_7$.

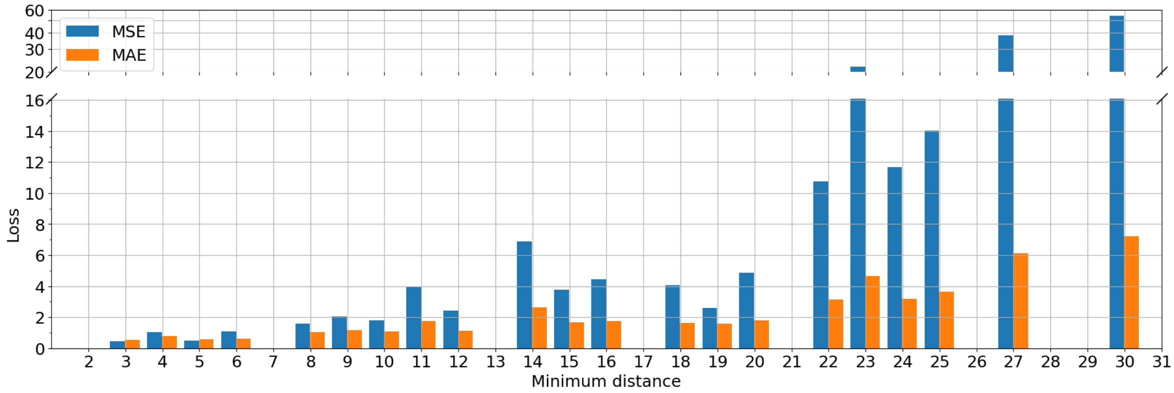


FIGURE 9. Losses on a test set of generalised toric codes over $\mathbb{F}_7$, collated by minimum Hamming distance, using an LSTM model.

generator matrices. The target outputs d in the dataset were rescaled to lie in the range $0 \leq \widetilde{d} \leq 1$ using the formula $\widetilde{d} = (d - \mu)/\sigma$, where $\mu = (30 + 3)/2 = 16$ and $\sigma = (30 - 3)/2 = 13.5$. The model consisted of two layers of stacked LSTMs. 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 used are given in Table 9.

On the training set, the model's MAE was 2.72 and the accuracy within the tolerance $d \pm 3$ was 93.0%. On the test set, the model's MAE was 3.02 and the accuracy within the tolerance $d \pm 3$ was 91.2%. The per-minimum-distance metrics are illustrated in Figure 9. The model performed better for smaller minimum Hamming distances, and worse for larger distances. This model serves as a proof of concept that it is possible to learn to predict the minimum Hamming distance for a linear code to reasonable accuracy, and that the model is able to generalise to unseen data.

Recall that the goal is to use the model as an estimator of the minimum Hamming distance, providing a fitness function for a genetic algorithm (GA). During our experiments with GAs, we observed that the final population after a GA run was somewhat noisy: the range between the smallest and the largest minimum Hamming distance typically varied from 3 to 10 or more. As a consequence, the model does not need to be 100% accurate in order to be efficient in this context; a MAE error within a reasonable range would still be sufficient for practical use.

Toric transformer. Recent advances in natural language processing (NLP) are driven by the transformer architecture [58], which has demonstrated high effectiveness on various sequence processing tasks. As discussed in the section 'Machine learning the minimum Hamming distance', the problem of predicting the minimum Hamming distance from the generator matrix can be viewed as a sequence classification task; it is therefore reasonable to attempt to modify the transformer to this setting. Compared to NLP transformers, there are several architectural changes we need to make:

(1) input encoding of the field elements (for $\mathbb{F}_8$);
(2) masking to prevent padding from affecting results;
(3) summarisation along time to produce a sequence length independent output.

We discuss these changes below when we describe the architecture in more detail. The general structure of the model is depicted in Figure 5, and is shared by both the model for codes over $\mathbb{F}_7$, and the model for codes over $\mathbb{F}_8$.

Input. The input of the model is composed of a code tensor and a length vector. The code tensor is a `PyTorch` [48] tensor containing a batch of generator matrices, padded with zeros to standardise the length. This tensor had three dimensions, denoted by (B, T, C), where:

(1) B is the *batch dimension*, i.e. the number of examples in a batch;
(2) T is the *time dimension*, corresponding to the maximum number of rows in the generator matrices, i.e. $(q-1)^2$ (for individual generator matrices, this dimension has a size equal to the dimension k of the linear code, but when padded and packed in the code tensor, it is equal to $(q-1)^2$);
(3) C is the *number of channels*, corresponding to the length of each row in the generator matrix, i.e. $(q-1)^2$.

The length vector stores the actual number of rows for each generator matrix in the batch. It can be used to distinguish between the rows of the generator matrix and any additional padding.

Input embedding. The input sequence – 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 matrix are already in a vector form with entries of the vectors being elements of $\mathbb{F}_q$. Elements of $\mathbb{F}_7$ can naturally be regarded as integers in the set $\{0, \ldots, 6\}$, and thus the rows can be used as an input as is. This naïve approach, however, does not work for $\mathbb{F}_8$. Instead fix a primitive element $\xi \in \mathbb{F}_8^*$. Then the elements of $\mathbb{F}_8$ can be written as $\{0, 1, \xi, \xi^2, \ldots, \xi^6\}$. Unfortunately these elements are not linearly independent, and although one-hot encoding was found to be efficient in the $\mathbb{F}_7$ case, it became unreasonable over $\mathbb{F}_8$. The alternative we used was to include an embedding layer with learnable representation; this is less efficient compared with a one-hot representation, in the sense that training to the same level of loss generally takes more epochs.

Attention blocks. The attention mechanism in the transformer architecture and embedding align naturally with the structure of the problem of predicting minimum Hamming distances. After passing the rows of the generator matrix to the model, it will first represent them as 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 the standard transformer architecture, the number of dimensions of the input – which in our model is of shape (B, T, C) – is preserved at each stage. But for classification tasks, a single fixed-size vector per sequence is needed, and so we must eliminate the time dimension T in the output. This is, we must introduce a pooling layer. There are many variants in the literature, with the simplest being:

(1) *BERT-like pooling.* Introduce the special classification token `[CLS]` in the input, and then use the output at `[CLS]` for classification.
(2) *Average pooling.* Take the average of all output along the T dimension and use this for classification.
(3) *Attention pooling.* Applying an attention mechanism to compute a weighted sum of sequence elements.

It was shown in [55] that the later two are somewhat better than the first approach, and so we use them in our model.

We used average pooling in the $\mathbb{F}_7$ 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, and so attention pooling was applied. The idea here is to perform a transformation $v \mapsto \text{softmax}(v^T M)v$ with a learnable square matrix M. This can reproduce the mean when M is proportional to the identity matrix and, therefore, should provide results that are no worse than the simple mean.

Output. Next, 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 applied to the pooled output. Finally, a softmax function was applied to obtain a vector $\vec{p}$ of probabilities of classes. These probabilities were then used to compute the expectation of the minimum Hamming distance via $\mathbb{E}[d] = \vec{p} \cdot \vec{i}$, where $\vec{i}$ is the vector of class labels (i.e. target minimum Hamming distances). Since we were working with padded data, we also had to add masking to prevent padding elements from influencing the results in the attention blocks and pooling layers.

Loss functions. Since we compute the (predicted) expected minimum Hamming distance, we could use the mean square error loss between the expected distance and the true distance. However, this loss would discard information about the individual probabilities of the classes. Instead, one should use either Wasserstein loss or cross-entropy loss.

Wasserstein loss [22] has applications within Generative Adversarial Networks (GANs). It measures the distance between probability distributions using some specified metric (we chose the Euclidean distance in experiments). We use this for codes over $\mathbb{F}_7$. *Cross-entropy loss* [45] 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. Our experiments demonstrated a superior performance for cross-entropy loss for codes over $\mathbb{F}_8$.

Hyperparameters. Tables 10 and 11 list the hyperparameters of transformer models for generalised toric codes over $\mathbb{F}_7$ and $\mathbb{F}_8$, respectively.

Hyperparameter	**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	Optimiser	AdamW
	Learning Rate	$3 \cdot 10^{-4}$
	Epochs	22

Table 10. The transformer architecture and training hyperparameters used for predicting minimum Hamming distances for generalised toric codes over $\mathbb{F}_7$.

Improvements. We used the models described above to search for champion codes using a GA. We started our experiments with the model for generalised toric codes over $\mathbb{F}_7$, iteratively improving both the 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 integers $0, \ldots, 6$ (and flattened afterwards). As a result, each input vector, that is, each row of the generator matrix, of length 36 is replaced by a vector of length 252.

As described in the section 'Datasets of generalised toric codes' above, 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 codes over $\mathbb{F}_7$, we used the entire dataset, which we partitioned into training and test sets. But unless we oversample some classes by a factor of thousands, the classes will be extremely unbalanced. To overcome this problem, we introduced a two-stage training procedure. First, in the pre-training stage, we used classes with many representatives (more than 10,000 in the case of codes

Hyperparameter	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	Optimiser	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 11. The transformer architecture and training hyperparameters used for predicting minimum Hamming distances for generalised toric codes over $\mathbb{F}_8$.

over $\mathbb{F}_7$). This allows the model to learn general patterns useful for minimum Hamming distance prediction. Second, in the training stage, we aimed to have all available classes represented, and so we down-sampled classes with many representatives (down to 500 elements), and upsampled classes with too few representatives (up to 500 elements), and trained the pre-trained model on the resulting dataset. This second training stage uses a relatively small dataset (14,000 codes over $\mathbb{F}_7$) and so the model is prone to overfitting: early stopping should be used.

Overall, this showed excellent results for codes over $\mathbb{F}_7$; the next iteration of the model (not used with GA) achieving cross-entropy loss of 0.61 on the pre-training dataset and 1.26 on the training dataset, and accuracy of 84.8% (within a tolerance of $d \pm 3$) on the training dataset and 90.3% on the test set. The counter-intuitive higher accuracy on the test than than on the training set could potentially be explained by the fact that the training set has oversampled classes.

Applying this methodology to codes over $\mathbb{F}_8$ demonstrated less impressive results. Already at the pre-training stage, with plenty of data, the model was prone to overfitting after just 120 epochs. In general, there are approaches to overcoming this problem: increase the dataset size or decrease the number of parameters in the model. Knowing that the task of predicting minimum Hamming distance for generalised toric codes over $\mathbb{F}_8$ is complicated, we chose not to decrease the model size. One might hope to improve the results by increasing the size of the dataset, however data generation remains challenging due to the high computational cost of the BZ algorithm.

Data availability. Our datasets are available from GitHub [34].

Code availability. All code required to replicate the results in this paper is available from GitHub [34] under an MIT license.

Acknowledgements. AK is funded by EPSRC Fellowship EP/N022513/1. QL is funded by a Mitchell Fund award from Trinity College, Oxford.

References

[1] Josh Abramson, Jonas Adler, Jack Dunger, Richard Evans, Tim Green, Alexander Pritzel, Olaf Ronneberger, Lindsay Willmore, Andrew J. Ballard, Joshua Bambrick, Sebastian W. Bodenstein, David A. Evans, Chia-Chun Hung, Michael O'Neill, David Reiman, Kathryn Tunyasuvunakool, Zachary Wu, Akvilė Žemgulytė, Eirini Arvaniti, Charles Beattie, Ottavia Bertolli, Alex Bridgland, Alexey Cherepanov, Miles Congreve, Alexander I. Cowen-Rivers, Andrew Cowie, Michael Figurnov, Fabian B. Fuchs, Hannah Gladman, Rishub Jain, Yousuf A. Khan, Caroline M. R. Low, Kuba Perlin, Anna Potapenko, Pascal Savy, Sukhdeep Singh, Adrian Stecula, Ashok Thillaisundaram, Catherine Tong, Sergei Yakneen, Ellen D. Zhong, Michal Zielinski, Augustin Žídek, Victor Bapst, Pushmeet Kohli, Max Jaderberg, Demis Hassabis, and John M. Jumper. Accurate structure prediction of biomolecular interactions with AlphaFold 3. *Nature*, 630(8016):493—500, 2024. doi:10.1038/s41586-024-07487-w.

[2] James E. Amaya, April J. Harry, and Brian M. Vega. A systematic census of generalized toric codes over $\mathbb{F}_4$, $\mathbb{F}_5$ and $\mathbb{F}_{16}$. Technical report, MSRI-UP, 2009. https://legacy.slmath.org/attachments/up/MSRI-UP%202009%20Technical%20Reports%20for%20Web.pdf#page=12.

[3] Leo Ardon. Reinforcement learning to solve NP-hard problems: an application to the CVRP. arXiv:2201.05393 [cs.AI], 2022.

[4] Erdal Arı kan. Channel polarization: a method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Trans. Inform. Theory*, 55(7):3051–3073, 2009. doi:10.1109/TIT.2009.2021379.

[5] Francesco Barbieri, Jose Camacho-Collados, Luis Espinosa Anke, and Leonardo Neves. TweetEval: Unified benchmark and comparative evaluation for tweet classification. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1644–1650, Online, November 2020. Association for Computational Linguistics. doi:10.18653/v1/2020.findings-emnlp.148.

[6] Peter Beelen and Diego Ruano. The order bound for toric codes. In Maria Bras-Amorós and Tom Høholdt, editors, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 1–10, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. doi:10.1007/978-3-642-02181-7_1.

[7] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828, 2013. doi:10.1109/TPAMI.2013.50.

[8] Per Berglund, Yang-Hui He, Elli Heyes, Edward Hirst, Vishnu Jejjala, and Andre Lukas. New Calabi–Yau manifolds from genetic algorithms. *Phys. Lett. B*, 850:Paper No. 138504, 10, 2024. doi:10.1016/j.physletb.2024.138504.

[9] R. C. Bose and D. K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3:68–79, 1960. doi:10.1016/S0019-9958(60)90287-4.

[10] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. doi:10.1006/jsco.1996.0125.

[11] Stefka Bouyuklieva and Iliya Bouyukliev. An extension of the brouwer–zimmermann algorithm for calculating the minimum weight of a linear code. *Mathematics*, 9(19), 2021. doi:10.3390/math9192354.

[12] Gavin Brown and Alexander M. Kasprzyk. Seven new champion linear codes. *LMS J. Comput. Math.*, 16:109–117, 2013. doi:10.1112/S1461157013000041.

[13] Gavin Brown and Alexander M. Kasprzyk. Small polygons and toric codes. *J. Symbolic Comput.*, 51:55–62, 2013. doi:10.1016/j.jsc.2012.07.001.

[14] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, 2020. https://papers.nips.cc/paper_files/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[15] Alejandro Carbonara, Juan Pablo Murillo, and Abner Ortiz. A census of two dimensional toric codes over Galois fields of sizes 7, 8 and 9. Technical report, MSRI-UP, 2009. https://legacy.slmath.org/attachments/up/MSRI-UP%202009%20Technical%20Reports%20for%20Web.pdf#page=38.

[16] Tom Coates, Alexander M. Kasprzyk, and Sara Veneziale. Machine learning detects terminal singularities. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 67183–67194, 2023. https://proceedings.neurips.cc/paper_files/paper/2023/hash/d453490ada2b1991852f053fbd213a6a-Abstract-Conference.html.

[17] Tom Coates, Alexander M. Kasprzyk, and Sara Veneziale. Machine learning the dimension of a Fano variety. *Nat. Commun.*, 14(5526), 2023. doi:10.1038/s41467-023-41157-1.

[18] Alain Couvreur and Hugues Randriambololona. Algebraic geometry codes and some applications. In *Concise encyclopedia of coding theory*, pages 307–361. CRC Press, Boca Raton, FL, 2021. doi:10.1201/9781315147901.

[19] Alex Davies, Petar Veličković, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, Richard Tanburn, Peter Battaglia, Charles Blundell, András Juhász, Marc Lackenby, Geordie Williamson, Demis Hassabis, and Pushmeet Kohli. Advancing mathematics by guiding human intuition with AI. *Nature*, 600:70–74, 2021. doi:10.1038/s41586-021-04086-x.

[20] The Sage Developers, William Stein, David Joyner, David Kohel, John Cremona, and Burçin Eröcal. Sagemath, version 9.0, 2020. doi:10.5281/zenodo.4066866.

[21] Harold Fredricksen. Error correction for deep space network teletype circuits (Technical Report 32-1275). Technical report, 1968. https://ntrs.nasa.gov/api/citations/19680016272/downloads/19680016272.pdf.

[22] Charlie Frogner, Chiyuan Zhang, Hossein Mobahi, Mauricio Araya, and Tomaso A. Poggio. Learning with a Wasserstein loss. In C. Cortes, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29, pages 2053—2061, 2015. https://papers.nips.cc/paper_files/paper/2015/hash/a9eb812238f753132652ae09963a05e9-Abstract.html.

[23] Ahmed Fawzy Gad. PyGAD: An intuitive genetic algorithm Python library. *Multimed. Tools Appl.*, 83:58029–58042, 2024. doi:10.1007/s11042-023-17167-y.

[24] R. G. Gallager. Low-density parity-check codes. *IRE Trans.*, IT-8:21–28, 1962. doi:10.1109/tit.1962.1057683.

[25] V D Goppa. Algebraico–geometric codes. *Mathematics of the USSR-Izvestiya*, 21(1):75–91, 1983. doi:10.1070/IM1983v021n01ABEH001641.

[26] Markus Grassl. Bounds on the minimum distance of linear codes and quantum codes. Online, available at https://www.codetables.de.

[27] Markus Grassl. Searching for linear codes with large minimum distance. In *Discovering mathematics with Magma*, volume 19 of *Algorithms Comput. Math.*, pages 287–313. Springer, Berlin, 2006. doi:10.1007/978-3-540-37634-7_13.

[28] Sergei Gukov, James Halverson, and Fabian Ruehle. Rigor with machine learning from field theory to the Poincaré conjecture. *Nature Rev. Phys.*, 6(5):310–319, 2024. doi:10.1038/s42254-024-00709-0.

[29] Johan P. Hansen. Toric surfaces and error-correcting codes. In *Coding theory, cryptography and related areas (Guanajuato, 1998)*, pages 132–142. Springer, Berlin, 2000. doi:10.1007/978-3-642-57189-3_12.

[30] Johan P. Hansen. Toric varieties Hirzebruch surfaces and error-correcting codes. *Appl. Algebra Engrg. Comm. Comput.*, 13(4):289–300, 2002. doi:10.1007/s00200-002-0106-0.

[31] Baran Hashemi, Roderic Guigo Corominas, and Alessandro Giacchetto. Can transformers do enumerative geometry? In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, editors, *International Conference on Representation Learning*, volume 2025, pages 70047–70067, 2025. https://proceedings.iclr.cc/paper_files/paper/2025/hash/aee2f03ecb2b2c1ea55a43946b651cfd-Abstract-Conference.html.

[32] Yang-Hui He. Machine-learning the string landscape. *Phys. Lett. B*, 774:564–568, 2017. doi:10.1016/j.physletb.2017.10.024.

[33] Yang-Hui He. AI-driven research in pure mathematics and theoretical physics. *Nature Reviews Physics*, 6(9):546–553, 2024. doi:10.1038/s42254-024-00740-1.

[34] Yang-Hui He, Alexander M. Kasprzyk, Q Le, and Dmitrii. Riabchenko. Machine learning discovers new champion codes: source code and datasets, 2025. https://github.com/QTVLe/MachineLearningDiscoversNewChampionCodes.

[35] Jerold Heller and Irwin Jacobs. Viterbi decoding for satellite and space communication. *IEEE Transactions on Communication Technology*, 19(5):835–848, 1971. doi:10.1109/TCOM.1971.1090711.

[36] Fernando Hernando, Francisco D. Igual, and Gregorio Quintana-Ortí. 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(2):Art. 23, 28, 2019. doi:10.1145/3302389.

[37] Raymond Hill. *A first course in coding theory*. Oxford Applied Mathematics and Computing Science Series. The Clarendon Press, Oxford University Press, New York, 1986.

[38] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi:10.1162/neco.1997.9.8.1735.

[39] A. Hocquenghem. Codes correcteurs d'erreurs. *Chiffres*, 2:147–156, 1959.

[40] David Joyner. Toric codes over finite fields. *Appl. Algebra Engrg. Comm. Comput.*, 15(1):63–79, 2004. doi:10.1007/s00200-004-0152-x.

[41] Rajan Kadel, Krishna Paudel, Deepani B Guruge, and Sharly J Halder. Opportunities and challenges for error control schemes for wireless sensor networks: A review. *Electronics*, 9(3):504, 2020. doi:10.3390/electronics9030504.

[42] Andrej Karpathy. minGPT, 2022. https://github.com/karpathy/minGPT.

[43] John Little and Hal Schenck. Toric surface codes and Minkowski sums. *SIAM J. Discrete Math.*, 20(4):999–1014, 2006. doi:10.1137/050637054.

[44] John B. Little. Remarks on generalized toric codes. *Finite Fields Appl.*, 24:1–14, 2013. doi:10.1016/j.ffa.2013.05.004.

[45] Anqi Mao, Mehryar Mohri, and Yutao Zhong. Cross-entropy loss functions: Theoretical analysis and applications. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pages 23803–23828, 2023. https://proceedings.mlr.press/v202/mao23b.html.

[46] D. E. Muller. Application of Boolean algebra to switching circuit design and to error detection. *Transactions of the I.R.E. Professional Group on Electronic Computers*, EC-3(3):6–12, 1954. doi:10.1109/IREPGELC.1954.6499441.

[47] JL Pamart, E Lefranc, S Morin, G Balland, YC Chen, TM Kissell, and JL Miller. Forward error correction in a 5 Gbit/s 6400 km EDFA based system. *Electronics Letters*, 30(4):342–343, 1994. doi:10.1049/el:19940242.

[48] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. https://papers.neurips.cc/paper_files/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

[49] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[50] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *J. Soc. Indust. Appl. Math.*, 8:300–304, 1960. http://www.jstor.org/stable/2098968.

[51] Irving S. Reed. A class of multiple-error-correcting codes and the decoding scheme. *Trans. IRE*, (PGIT-, PGIT-4):38–49, 1954. doi:10.1109/TIT.1954.1057465.

[52] Diego Ruano. On the structure of generalized toric codes. *J. Symbolic Comput.*, 44(5):499–506, 2009. doi:10.1016/j.jsc.2007.07.018.

[53] Moshe Sipper, Weixuan Fu, Karuna Ahuja, and Jason H. Moore. Investigating the parameter space of evolutionary algorithms. *BioData Mining*, 11:2, 2018. doi:10.1186/s13040-018-0164-x.

[54] Ivan Soprunov and Jenya Soprunova. Toric surface codes and Minkowski length of polygons. *SIAM J. Discrete Math.*, 23(1):384–400, 2008. doi:10.1137/080716554.

[55] Yixuan Tang and Yi Yang. Pooling and attention: What are effective designs for llm-based embedding models? arXiv:2409.02727 [cs.CL], 2024.

[56] R. J. M. Vaessens, E. H. L. Aarts, and J. H. van Lint. Genetic algorithms in coding theory—a table for $A_3(n, d)$. *Discrete Appl. Math.*, 45(1):71–87, 1993. doi:10.1016/0166-218X(93)90141-A.

[57] Alexander Vardy. The intractability of computing the minimum distance of a code. *IEEE Trans. Inform. Theory*, 43(6):1757–1766, 1997. doi:10.1109/18.641542.

[58] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, 2017. https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[59] Yue Wu and Jesús A De Loera. Turning mathematics problems into games: Reinforcement learning and Gröbner bases together solve integer feasibility problems. arXiv:2208.12191 [cs.LG], 2022.

[60] Morris Yau, Nikolaos Karalias, Eric Lu, Jessica Xu, and Stefanie Jegelka. Are graph neural networks optimal approximation algorithms? In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 73124–73181, 2024. https://proceedings.neurips.cc/paper_files/paper/2024/hash/85db52cc08c5e00cfb1d216b1c85ba35-Abstract-Conference.html.

[61] Karl-Heinz Zimmermann. Integral Hecke modules, integral generalized Reed–Muller codes, and linear codes. Technical report, Techn. Univ. Hamburg-Harburg, 1996. http://hdl.handle.net/11420/9651.

London Institute for Mathematical Sciences, Royal Institution, London, W1S 4BS, UK
Department of Mathematics, City St George's, University of London, London, EC1V 0HB, UK
Merton College, University of Oxford, Oxford, OX1 4JD, UK
Email address: `yh@lims.ac.uk`

Mathematics Institute, University of Warwick, Coventry, CV4 7AL, UK
Email address: `alexander.kasprzyk@warwick.ac.uk`

Mathematical Institute, University of Oxford, Oxford, OX2 6GG, UK
Email address: `q.le@trinity.ox.ac.uk`

Department of Mathematics, City St George's, University of London, London, EC1V 0HB, UK
Email address: `Dmitrii.Riabchenko@citystgeorges.ac.uk`