AI for Code-based Cryptography

Mohamed Malhou[1,2][0009-0005-8412-4135], Ludovic Perret[3,2][0009-0005-7453-8705], and Kristin Lauter[1][0000-0002-1320-696X]

[1] FAIR at Meta {mmalhou, klauter}@meta.com
[2] Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
[3] EPITA, EPITA Research Lab (LRE), Le Kremlin-Bicêtre, France
ludovic.perret@epita.fr

Abstract. We introduce the use of machine learning in the cryptanalysis of code-based cryptography. Our focus is on distinguishing problems related to the security of NIST round-4 McEliece-like cryptosystems, particularly for Goppa codes used in `ClassicMcEliece` and Quasi-Cyclic Moderate Density Parity-Check (QC-MDPC) codes used in `BIKE`. We present `DeepDistinguisher`, a new algorithm that trains a transformer to distinguish structured codes from random linear codes. The results show that the new distinguisher achieves high accuracy in distinguishing Goppa codes, suggesting that their structure may be more recognizable by AI models. Our approach outperforms traditional attacks for distinguishing Goppa codes in certain settings and generalizes to longer code lengths without further training using a puncturing technique. We also present the first distinguishing results for MDPC and QC-MDPC codes.

Keywords: Classic McEliece · Goppa Codes · QC-MDPC · Code Distinguishability · Deep Learning · Transformers

1 Introduction

In recent years, the cryptographic community has been actively preparing for the cyber-security challenges arising from the impending advent of cryptographically relevant quantum computers. To address this quantum threat, the National Institute of Standards & Technology (NIST) started a multi-stage standardization effort [11] to replace current number-theoretic-based cryptographic standards with a new generation of quantum-resistant algorithms. In 2024, NIST standardized a first set of post-quantum cryptography (PQC) standards, including the Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM, [39]), the Module-Lattice-Based Digital Signature Algorithm (ML-DSA, [38]) and the Stateless Hash-Based Digital Signature Algorithm (SLH-DSA, [40]).

The standardization of post-quantum cryptography is still ongoing, with NIST currently conducting a fourth round of evaluations to identify additional key encapsulation mechanisms (KEMs) [41]. Candidates that remain in the fourth round all belong to code-based cryptography [8,42,18], a family based on the algorithmic and NP-hardness of decoding random linear codes [6,18]. In particular, two candidates follow the general framework of the McEliece cryptosystem [35]: ClassicMcEliece [1] that uses binary Goppa codes as initially proposed by Robert McEliece in 1978, and BIKE [37,2] a variant relying on Quasi-Cyclic Moderate Density Parity-Check (QC-MDPC) codes.

A fundamental question for these schemes, and post-quantum cryptography in general, is the hardness of the underlying algorithmic problems. This issue is both critical and highly challenging, as the security of standardized and candidate schemes for NIST has been intensively scrutinized in the past few years. Introducing any new cryptanalytic technique can be considered a notable achievement.

In this context, recent advances in Machine Learning (ML) provide a new paradigm to accelerate cryptanalysis. In particular, deep learning models – especially transformer-based architectures – have demonstrated remarkable success in pattern recognition, feature extraction, and automated discovery of hidden structures in high-dimensional data. In [49,32,33,50], the authors introduced the use of transformers to attack the Learning With Errors (LWE) problem [45], a central problem in post-quantum cryptography. The capability of these models to learn joint distributions of sequential data makes them a promising tool for identifying latent structures in algebraic constructions.

This paper presents a novel application of ML techniques to assess the security of code-based public-key cryptosystems. Specifically, we consider the problem of distinguishing structured public codes (e.g., Goppa or QC-MDPC) from random codes. To do so, we design a supervised learning framework on finite field data and introduce a transformer-based algorithm, DeepDistinguisher, designed to classify structured codes more effectively. Our work does not directly impact the security of Classic McEliece; however, we believe that our findings will inspire researchers to further explore this problem. To our knowledge, this is the first work that applies ML to a core hardness assumption in code-based cryptography rather than to side-channel leakage.

We validate our DeepDistinguisher via extensive experiments across a range of code parameters. For Goppa codes, we empirically demonstrate that DeepDistinguisher can classify structured codes from random with high accuracy, even outperforming the most recent approaches [43,20,13] for some specific parameters.

Organization of the paper

The paper is organized as follows. We begin with a discussion of prior research relevant to our approach in 1 which appears as part of this introduction. Then, sections 2 and 3 provide the necessary background and definitions for understanding our work.
Our DeepDistinguisher is detailed in Section 4 where we describe the training framework, data generation process, and evaluation strategies. We highlight the importance of the choice of deep learning architecture and data representation.

Section 5 presents our experimental results, demonstrating that our model achieves high classification accuracy and outperforms traditional algebraic distinguishers such as FGOPT [20], CMT [13], and syzygy [43] on the toy parameter settings proposed in [43].
Additionally, we present distinguishing results on ternary Goppa codes, and certain binary alternant codes, but also the first specific distinguishing results on (QC) MDPC codes. Although our experimental results are limited, our findings suggest that distinguishing these codes is easier than finding low-weight codewords, the approach taken in [37].

Finally, in Section 6, we introduce a more challenging problem: given a public generator matrix of a Goppa code with missing entries, recover the missing values such that the outcome is a valid Goppa code. This is a harder problem than distinguishing and seems impossible without the knowledge of some information about the private key. Our model successfully recovers these missing values, demonstrating that the structure of Goppa codes can be learned and exploited by AI and that our `DeepDistinguisher` is not simply the statistical distinguisher described in Appendix A. The code has been open sourced for the community.[4]

Related work

AI, and more specifically ML, is becoming a powerful approach in cryptanalysis, with a growing body of research demonstrating that neural networks can detect patterns. For instance, Gohr [23] showed that deep residual neural networks can serve as exceptionally strong differential distinguishers—outperforming classical methods on round-reduced Speck32/64 with very few chosen plaintexts.
In post-quantum cryptography, a first generation of ML techniques were used in [26] to attack group-based cryptosystems. More recently, the SALSA papers [49,32,33,50] leveraged recent advances in ML —- in particular the introduction of transformers [48] – to solve Learning with Errors (LWE) problems. Lastly, [29] considers learning-based information-theoretic metrics, leveraging mutual information estimation and binary classification to evaluate the security of cryptographic schemes under chosen-plaintext attacks (`IND-CPA`). The study demonstrates that neural networks can efficiently identify cryptosystems that are not `IND-CPA` secure by modeling the distinguishability of ciphertexts as a classification task.

In this paper, we present the first ML-based attack, `DeepDistinguisher`, on the distinguishing problem arising in the security of McEliece-like cryptosystems. In particular, we consider the *Goppa Code Distinguishing* (GD) problem [12]; probably the most famous example of code distinguishing problem.

*Problem 1 (Goppa Code Distinguishing (*GD*) problem). Given a generator matrix* $\mathbf{G} \in \mathcal{M}_{k\times n}(\mathbb{F}_q)$ *of a* $[n,k]_q$ *linear code, the Goppa Code Distinguishing (*GD*) problem asks to decide if* $\mathbf{G}$ *is the generator matrix of a Goppa code or a randomly drawn matrix.*

[4] Code is available at github.com/facebookresearch/ai4code-cryptanalysis

The GD problem was formally introduced in [12] and was initially believed to be hard. Thus, it served as an assumption for reducing the security of the McEliece cryptosystem to the problem of decoding random linear codes [17]. Our understanding of the hardness of GD has significantly shifted in the past ten years, culminating with the so-called *syzygy distinguisher* [43] that now solves GD for a broad range of parameters with a complexity asymptotically faster than the best generic decoding algorithms.

The syzygy distinguisher, as well as improved results on GD such as [14,13], is built on the polynomial-time distinguisher presented by Faugère, Gautier, Otmani, Perret, Tillich (FGOPT) [20]. The core idea behind the FGOPT distinguisher is to analyze the behavior of the Gröbner basis computation [10,9] of an algebraic system associated to McEliece's public key. This computation behaves differently if the algebraic system is generated from a McEliece public key or from a randomly generated matrix. FGOPT described specific linear relations occurring in such computations due to the Goppa (or alternant) structure, leading to a polynomial-time distinguisher solving GD for codes whose rate $R = \frac{k}{n}$ is close to 1. Since this result, a major open question has been how to extend the distinguishing rate.

At Asiacrypt'23, Couvreur, Mora and Tillich (CMT, [13]) finally demonstrated that the approach from [20] can be improved. CMT introduced a new algebraic modeling method and leveraged more general *algebraic relations*, known as syzygies, arising in the Gröbner basis computations. While FGOPT can distinguish codes with rates extremely close to 1, CMT pushed the boundaries of success to distinguishing rates in the range $[\frac{2}{3}, 1]$.

A central contribution of the latest distinguisher [43] is to precisely predict the syzygies arising at any step of the Gröbner computations. In particular, such a distinguisher can solve GD for a broad range of parameters with complexity which is sub-exponential in the error-correcting capacity. Asymptotically, this removes the limitation on the rate for the syzygy distinguisher. For fixed parameters, however, the situation is different. Remark that, unlike FGOPT, the CMT and syzygy distinguishers are not polynomial-time algorithms. The rate remains a limiting factor, and certain code parameters cannot be distinguished by either the syzygy or CMT distinguishers due to fundamental theoretical limitations and/or computational complexity constraints.

The code distinguishing problem for Quasi-Cyclic (QC) and general Moderate Density Parity-Check (MDPC) was formally introduced in [37]. To the best of our knowledge, no dedicated technique exists for distinguishing such codes. The only known approach, described in [37], relies on finding low-weight codewords in the public code – a problem equivalent to message recovery. This led the authors of [37] to introduce an assumption about the (exponential) hardness of distinguishing MDPC codes.

We emphasize that the hardness of GD has no direct impact on the security of McEliece yet. That is, there is currently no generic technique to mount an attack against McEliece using a distinguisher. However, recent results [5,13] demonstrated that techniques used for distinguishing alternant/Goppa codes can also be applied to attack a version of McEliece using generic alternant codes with high rates.

2 Preliminaries

2.1 Notations

Finite fields. We consider the finite field $\mathbb{F}_q$ of order q with q a prime power. For some integer $m > 0, \mathbb{F}_{q^m}$ is a field extension of $\mathbb{F}_q$ of degree m: $\mathbb{F}_{q^m} \cong \mathbb{F}_q[x]/(g(x)) \cong \mathbb{F}_q[\alpha]$ with α a root of an irreducible polynomial $g(x)$ of degree m. Any element $\beta \in \mathbb{F}_{q^m}$ can be naturally associated with its vector form in $\mathbb{F}_q$ as $(c_0, \ldots, c_{m-1}) \in \mathbb{F}_q^m$, where $\beta = \sum_{i=0}^{m-1} c_i \alpha^i$.

Vectors and matrices. We use lowercase letters to represent integers, while integer intervals are expressed as $[\![a;b]\!]$. Matrices are denoted by bold uppercase letters, and vectors by bold lowercase letters. For a vector $\mathbf{v}$, the notation v_i refers to its i-th component, and $\mathbf{v}^\top$ denotes its transpose. $\mathcal{M}_{k\times n}(\mathbb{F})$ will denote the set of $k \times n$ matrices with coefficients over a finite field $\mathbb{F}$.

For a matrix $\mathbf{A} \in \mathcal{M}_{k\times n}(\mathbb{F})$, the element in the i-th row and j-th column is denoted by a_{ij}. A sub-matrix of $\mathbf{A}$, specified by a set of rows $\mathcal{I}$ and a set of columns $\mathcal{J}$, is written as $\mathbf{A}[\mathcal{I}, \mathcal{J}]$. Additionally, a specific row or column of a matrix $\mathbf{A}$ is indicated by $\mathbf{A}[i, :]$ and $\mathbf{A}[:, j]$, respectively.

2.2 Basics of deep learning

Before presenting our approach, we first introduce the fundamental concepts of deep learning to provide the necessary background for a clear understanding of our methodology [24].

A *deep neural network* is a parametric family of functions $F_\theta \colon \mathcal{X} \to \mathcal{Y}$, where $\theta \in \mathbb{R}^p$ represents all trainable parameters (i.e., the entries of weight matrices and bias vectors) [24] and $\mathcal{X}$ and $\mathcal{Y}$ are measurable spaces. Concretely, for an input $x \in \mathcal{X}$, one may write:

$$F_\theta \colon x \;\mapsto\; W_d\, \sigma\big(\cdots \sigma(W_1 x + b_1) \cdots\big) + b_d,$$

with each W_i a weight matrix, b_i a bias vector, and σ a fixed nonlinearity such as ReLU (Rectified Linear Unit $x \to \max(0, x)$) applied component-wise. In this case, the trainable parameters of F are $\theta = \{W_1, b_1, \ldots, W_d, b_d\}$.

Training amounts to minimizing an empirical risk

$$\min_\theta \sum_i \ell\big(F_\theta;\, x_i,\, y_i\big),$$

where $\{(x_i, y_i)\}$ is a labeled dataset and ℓ is a chosen loss function. A common case is *binary classification*, where $\mathcal{Y} = \{0, 1\}$ and one trains F_θ to output probabilities in $[0, 1]$ by minimizing the *binary cross-entropy* loss [24]:

$$\sum_i \Big[-\, y_i \log\big(F_\theta(x_i)\big) \;-\; (1 - y_i) \log\big(1 - F_\theta(x_i)\big)\Big].$$

Parameters θ are typically updated via gradient-based algorithms (e.g., stochastic gradient descent) that converge to a θ^* that produces accurate predictions on new (held-out/validation) data. By the *universal approximation* theorem, sufficiently large networks can approximate wide classes of continuous functions on compact domains [16,28].

A *Transformer encoder* [48] is a deep sequence-to-sequence model that takes an ordered collection $\{x_1, \ldots, x_n\} \subseteq \mathcal{X}$—which may be text tokens, image patches, matrix rows, etc.—and outputs a sequence of embeddings $H_L \in \mathbb{R}^{n \times d}$. Each x_i is first embedded (or projected) into $h_{0,i} \in \mathbb{R}^d$. Then, each of the L layers applies *multi-head self-attention* and a small *feed-forward* sub-network, with residual connections and normalization. Formally, for layer ℓ, we form affine queries, keys, and values from $H_{\ell-1}$, compute

$$\mathrm{Att}(Q, K, V) = \mathrm{softmax}\Big(\tfrac{QK^\top}{\sqrt{d_k}}\Big) V$$

for each attention head, concatenate the head outputs, and project them. We update

$$H'_\ell = \mathrm{LayerNorm}\big(H_{\ell-1} + \mathrm{MultiHeadAtt}(H_{\ell-1})\big)$$
$$H_\ell = \mathrm{LayerNorm}\big(H'_\ell + \mathrm{FFN}(H'_\ell)\big)$$

All parameters (in the attention and feed-forward blocks) are trained end-to-end via gradient descent to yield a final encoder representation H_L. If the task is a classification, then we project the hidden state H_L into the output space using a trainable linear layer [48,3].

3 Coding Theory

3.1 Linear Codes and the Bounded Distance Decoding Problem

A $[k, n]_q$ *linear code* $\mathcal{C} \subseteq \mathbb{F}_q^n$ is a k-dimensional subspace of $\mathbb{F}_q^n$. The rate of $\mathcal{C}$ is defined as k/n and elements of $\mathcal{C}$ are called codewords. $\mathcal{C}$ can be specified by a full rank *generator matrix* $\mathbf{G} \in \mathcal{M}_{k \times n}(\mathbb{F}_q)$ such that $\mathcal{C} = \{\mathbf{m} \cdot \mathbf{G} \mid \mathbf{m} \in \mathbb{F}_q^k\}$. The *standard form* of $\mathbf{G}$ is $\mathbf{G} = [\mathbf{I}_k \mid \mathbf{A}]$, with $\mathbf{I}_k$ being the $k \times k$ identity matrix. Equivalently, $\mathcal{C}$ can be represented by *parity-check matrix* $\mathbf{H} \in \mathcal{M}_{(n-k) \times n}(\mathbb{F}_q)$ that satisfies $\mathbf{H} \cdot \mathbf{c}^\top = \mathbf{0}_{(n-k)}, \forall \mathbf{c} \in \mathcal{C}$, and its row space is the dual of $\mathcal{C}$.

We introduce below a general operation on codes that will be used to extend the range of applicability of `DeepDistinguisher`.

Definition 1 (Punctured Code). *Given a code $\mathcal{C} \subseteq \mathbb{F}^n$, and a subset $\mathcal{I} \subset [\![1; n]\!]$, the punctured code over $\mathcal{I}$ is defined as :*

$$\mathcal{P}_{\mathcal{I}}(\mathcal{C}) = \Big\{ (c_i)_{i \in [\![1;n]\!] \setminus \mathcal{I}} \;\big|\, \mathbf{c} \in \mathcal{C} \Big\}.$$

Code-based cryptography [8,42,18] is based on the intractability, i.e. NP-Hardness, of the Bounded Distance Decoding (BDD, [6]) problem:

*Problem 2 (Bounded Distance Decoding (*BDD*) problem). Given the generator matrix $\mathbf{G} \in \mathcal{M}_{k\times n}(\mathbb{F}_q)$ of an $[n,k]_q$ linear code, a target $\mathbf{c} \in \mathbb{F}_q^n$, and an integer $t > 0$, the* BDD *problem asks to find – if any – $\mathbf{m} \in \mathbb{F}_q^k$ such that:*

$$w_H(\mathbf{c} - \mathbf{m} \cdot \mathbf{G}) \leq t,$$

with w_H being the Hamming weight of the vector, i.e. the number of its non-zero coordinates.

Solving BDD for random codes, i.e. random generator matrices $\mathbf{G} \in \mathcal{M}_{k\times n}(\mathbb{F}_q)$, is a long-standing problem whose most effective algorithms are all computationally intractable [47,7,18].

3.2 McEliece Framework and Code Distinguishing Problem

The McEliece cryptosystem [35] is certainly the most popular code-based public-key cryptosystem. In particular, round-4 NIST candidates `ClassicMcEliece` [1] (which is based on the Niederreiter variant) and `BIKE` [2] follow the general framework described below.

- **Secret-Key.** A structured generator matrix $\mathbf{G}_s \in \mathcal{M}_{k\times n}(\mathbb{F}_q)$ of a $[n,k]_q$ linear code with a known decoding algorithm.
- **Public-Key.** A scrambled generator matrix $\mathbf{G} \in \mathcal{M}_{k\times n}(\mathbb{F}_q)$ derived from the secret-key $\mathbf{G}_s$.
- **Encryption.** Given a message $\mathbf{m} \in \mathbb{F}_q^k$, the ciphertext is computed as $\mathbf{c} = \mathbf{m} \cdot \mathbf{G} + \mathbf{e} \in \mathbb{F}_q^n$, where $\mathbf{e} \in \mathbb{F}_q^n$ is an error vector of small Hamming weight.
- **Decryption.** Given a ciphertext $\mathbf{c} \in \mathbb{F}_q^n$, the receiver applies the code's decoding algorithm to recover the message $\mathbf{m}$.

From this description, it is clear that the security of McEliece (message-recovery) relies on the hardness of BDD. In addition, it is natural to introduce a general distinguishability problem for a structured family of linear codes $\mathscr{F}$.

*Problem 3 (Code Distinguishability (*CD*) problem). Given a generator matrix $\mathbf{G} \in \mathcal{M}_{k\times n}(\mathbb{F}_q)$ of an $[n,k]_q$ linear code, the* CD *problem asks to decide if $\mathbf{G}$ is the generator matrix of an $\mathscr{F}$-code or randomly drawn.*

In this paper, $\mathscr{F}$ includes Goppa or alternant ($\mathscr{G}, \mathscr{A}$) codes as well as MDPC and QC-MDPC ($\mathscr{M}, \mathscr{Q} \cap \mathscr{M}$) codes.

3.3 Alternant and Goppa Codes

The family of codes used in `ClassicMcEliece` can be conveniently described by introducing Generalized Reed-Solomon codes.

Definition 2 (Generalized Reed-Solomon Code, [44]). *Let* $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n) \in \mathbb{F}_q^n$ *be an* n*-tuple of distinct elements in* $\mathbb{F}_q$*, called a support, and* $\boldsymbol{\beta} = (\beta_1, \dots, \beta_n) \in (\mathbb{F}_q^*)^n$ *an* n*-tuple of nonzero elements in* $\mathbb{F}_q$*, called multiplier. The Generalized Reed-Solomon code of length* n *and dimension* t*, denoted by* $\mathrm{GRS}_{q,n,t}(\boldsymbol{\alpha}, \boldsymbol{\beta})$*, is defined as:*

$$\mathrm{GRS}_{q,n,t}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \{(\beta_1 f(\alpha_1), \dots, \beta_n f(\alpha_n)) \mid f \in \mathbb{F}_q[x], \deg(f) < t\}.$$

Remark that the following weighted Vandermonde matrix is a generator matrix of the GRS code.

$$\mathbf{V}_t[\boldsymbol{\alpha}, \boldsymbol{\beta}] = \begin{pmatrix} \beta_1 & \beta_2 & \cdots & \beta_n \\ \beta_1\alpha_1 & \beta_2\alpha_2 & \cdots & \beta_n\alpha_n \\ \vdots & \vdots & \ddots & \vdots \\ \beta_1\alpha_1^{t-1} & \beta_2\alpha_2^{t-1} & \cdots & \beta_n\alpha_n^{t-1} \end{pmatrix}.$$

Alternant codes can be viewed as subfield subcodes of GRS codes.

Definition 3 (Alternant Code, [34]). *Let* $\boldsymbol{\alpha} \in \mathbb{F}_{q^m}^n$ *be a support and* $\boldsymbol{\beta} \in (\mathbb{F}_{q^m}^*)^n$ *be a multiplier as in Definition 2 such that* $n \le q^m$*. The alternant code of degree* t*, denoted by* $\mathcal{A}_{q,n,m,t}(\boldsymbol{\alpha}, \boldsymbol{\beta})$*, is given by:*

$$\mathcal{A}_{q,n,m,t}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathrm{GRS}_{q^m,n,t}(\boldsymbol{\alpha}, \boldsymbol{\beta})^{\perp} \cap \mathbb{F}_q^n.$$

$\mathcal{A}_{q,n,m,t}(\boldsymbol{\alpha}, \boldsymbol{\beta})$ *is* $[n, k \ge n - mt]_q$ *linear code.*

Once the support and multiplier vectors are known, alternant codes of degree t can be decoded in polynomial-time up to errors with Hamming weight $t/2$ [34, Ch. 12]. McEliece cryptosystem relies on a sub-class of alternant codes.

Definition 4 (Goppa Code, [25]). *Let* $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n) \in \mathbb{F}_{q^m}^n$ *be a support and* $g(x) \in \mathbb{F}_{q^m}[x]$ *be a degree* t *square-free polynomial, called a Goppa polynomial, such that* $g(\alpha_i) \neq 0, \forall\ 1 \le i \le n$*. The Goppa code, denoted* $\mathcal{G}_{q,n,m,t}(\boldsymbol{\alpha}, g)$*, is defined as follows:*

$$\mathcal{G}_{q,n,m,t}(\boldsymbol{\alpha}, g) = \mathcal{A}_{q,n,m,t}\left(\boldsymbol{\alpha}, \frac{1}{g(\boldsymbol{\alpha})}\right).$$

$\mathcal{G}_{q,n,m,t}(\boldsymbol{\alpha}, g)$ *is an* $[n, k \ge n - mt]_q$ *linear code.*

Goppa codes, viewed as alternant codes, naturally inherit a decoding algorithm that corrects up to $t/2$ errors. For binary Goppa codes ($q = 2$), we can improve this bound to correct twice as many errors in polynomial-time.

3.4 Codes with Sparse Parity-Check Matrices

`BIKE` [37,2] relies on linear codes described by compact and sparse matrices.

Definition 5 (Moderate Density Parity-Check codes, [37]). *An* (n, k, w)-MDPC *code is a linear code of length* n*, co-dimension* k *admitting a parity check matrix with constant row weight* w *which scales in* $O(\sqrt{n \log n})$*.*

BIKE adds a structure to MDPC codes allowing to decrease the size of the public-key.

Definition 6 (Quasi-cyclic codes, [37]). *An $[n, k]_q$-linear code is Quasi-Cyclic* (QC) *if there is some integer ℓ with $n = \ell n_0$ such that every cyclic shift of a codeword by ℓ places is again a codeword.*

3.5 Solving the Code Distinguishing Problem

A well-studied example of a code distinguishing problem occurs when the family $\mathscr{F}$ is restricted to Goppa or alternant codes (section 3.3). This corresponds to the classical McEliece scheme and the Goppa Code Distinguishing (GD) problem.

The first efficient algorithm for solving this problem, FGOPT [20], relies critically on the code rate k/n. In [20,13], the authors precisely characterize the range of parameters for which FGOPT can distinguish Goppa codes in polynomial time.

Definition 7 (Square–distinguishable Goppa code). *A Goppa code $\mathcal{G}(\boldsymbol{\alpha}, g)$, with $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_{q^m}$ a support and $g(x) \in \mathbb{F}_{q^m}[x]$ a Goppa polynomial of degree t, is said to be square–distinguishable if:*

$$n > \binom{tm+1}{2} - \frac{m}{2}(t-1)(t-2), \qquad \textit{when } \; t < q-1 \tag{1}$$

$$n > \binom{tm+1}{2} - \frac{m}{2}t\left((2e_{\mathcal{G}}+1)t - 2(q-1)q^{e_{\mathcal{G}}-1} - 1\right), \; \textit{otherwise}, \tag{2}$$

where $e_{\mathcal{G}} = \min\{i \in \mathbb{N} \mid t \le (q-1)^2 q^i\} + 1 = \left\lceil \log_q\left(\frac{t}{(q-1)^2}\right)\right\rceil + 1.$

Note that similar results can be derived for alternant or binary Goppa codes.

In [13], Couvreur, Mora and Tillich (CMT) extended the concept of distinguishable codes by introducing a new class called d-distinguishable codes. This concept is based on invariants related to the Hilbert function, a fundamental tool from commutative algebra [15], commonly used to assess the complexity of Gröbner basis computations [10,9]. In particular, it applies to Pfaffian ideals, i.e. ideals generated by symbolic minors of skew-matrices [36,19], modeling specific relations of alternant and Goppa codes.

Definition 8 (d-distinguishable, simplified from [13]). *Let $\mathcal{C}$ be an $[n, tm]_{\mathbb{F}_{q^m}}$ linear code, $\mathcal{P}_2^+(\mathcal{C})$ be the Pfaffian ideal associated to $\mathcal{C}$ [13, Sec. 5.2] and $\mathrm{HF}_{\mathcal{P}_2^+(\mathcal{C})}$ be the corresponding Hilbert function. $\mathcal{C}$ is said to be d-distinguishable from a generic $[n, tm]$ linear code over $\mathbb{F}_{q^m}$ when the following holds:*

$$\mathrm{HF}_{\mathcal{P}_2^+(\mathcal{C})}(d) \neq \max\left(0, \sum_{i=0}^{d} \frac{(-1)^i}{tm+d-i-1}\binom{n-tm}{i}\binom{tm+d-i-1}{d-i+1}\binom{tm+d-i-1}{d-i}\right)$$

1-distinguishable codes correspond to square–distinguishable Goppa codes (Definition 7). In [13], the authors demonstrated that d-distinguishability, for $d > 1$, allows to distinguish a broader family of codes than FGOPT, albeit at a higher computational cost. In particular, the complexity of the CMT distinguisher is bounded from above by:

$$\mathcal{O}\left(\left(\binom{tm}{2} - k + 1\right)\binom{\binom{tm}{2} + d_{\text{reg}} - 1}{d_{\text{reg}}}^{\omega}\right), \tag{3}$$

where $2 \leq \omega < 3$ is a feasible linear algebra constant, and d_{reg} is the degree of regularity [4], i.e. the maximum degree reached in the computation of (degree-based) Gröbner basis of the Pfaffian ideal $\mathcal{P}_2^+(\mathcal{C})$. In [13], the authors conjecture that d_{reg} behaves asymptotically as $d_{\text{reg}} \sim c\frac{(tm)^2}{n-tm}$, where c is a constant close to $\frac{1}{4}$. These lead to a new algorithm that distinguishes codes with a rate in the range $[2/3, 1]$. Its complexity interpolates between polynomial-time (square–distinguishable Goppa codes) and super-exponential in the error-correcting capability of the code for constant rates.

The syzygy distinguisher [43] includes and extends previous results. It refines the algebraic modeling from CMT and conducts a more precise analysis of the syzygies occurring during a Gröbner computation. The dimension of these syzygies are related to so-called Betti numbers that depend on the structure of the code considered. These allow the author to present a new distinguisher that is asymptomatically independent of the rate. Its complexity is bounded from above by

$$\kappa = q^{\left(\omega\frac{R^2}{1-R}+o(1)\right)\frac{(\log_q \log_q(n))^3}{(\log_q(n))^2}n},$$

where R is the rate of the dual code $R = mt/n$, and ω is the linear algebra exponent. The algorithm is not polynomial-time, but remains sub-exponential in the error-correcting capacity.

4 A Transformer-Based Algorithm for Code Distinguishing

In this section, we introduce a novel and natural method for code distinguishing based on deep learning. The motivation behind this approach is that a deep learning model, trained to classify samples from different families of codes, can potentially identify patterns revealed by a public generator matrix. Unlike classical approaches that rely on predefined heuristics or algebraic properties, a deep learning model can adaptively discover hidden structural differences between code families if any.

4.1 Deep Distinguisher

Let $\mathscr{F} \in \{\mathscr{G}, \mathscr{A}, \mathscr{M}, \mathscr{Q}\cap\mathscr{M}\}$ denote the code family of interest (Goppa, alternant, MDPC, and QC-MDPC codes), at a high level, we aim to learn a parametric classifier

$$D_{\boldsymbol{\theta}} :\ \mathbb{F}_q^{k\times n} \longrightarrow\ [0,1],$$

where $D_{\boldsymbol{\theta}}$ is a function parameterized by $\boldsymbol{\theta} \in \mathbb{R}^{d_{\text{model}}}$ and $D_{\boldsymbol{\theta}}(\mathbf{G})$ estimates the probability that the public generator matrix $\mathbf{G}$ spans a code in the target family $\mathscr{F}$. During training, we sample labeled pairs $\big(\mathbf{G}^{(i)}, y^{(i)}\big)$ with

$$y^{(i)} = \begin{cases} 1, & \text{if } \mathbf{G}^{(i)} \text{ generates a code in } \mathscr{F}, \\ 0, & \text{otherwise,} \end{cases}$$

and minimize the empirical cross-entropy loss

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{N} \log\Big[\prod_{i=1}^{N} D_{\boldsymbol{\theta}}(\mathbf{G}^{(i)})^{y^{(i)}} \big(1 - D_{\boldsymbol{\theta}}(\mathbf{G}^{(i)})\big)^{1-y^{(i)}}\Big].$$

At inference time, we apply a threshold $\tau \in (0, 1)$ - usually $\frac{1}{2}$ - to obtain a binary decision rule:

$$\widehat{b}(\mathbf{G}_{\text{test}}) \;=\; \begin{cases} 1, & D_{\boldsymbol{\theta}}(\mathbf{G}_{\text{test}}) \;\geq\; \tau, \\ 0, & D_{\boldsymbol{\theta}}(\mathbf{G}_{\text{test}}) \;<\; \tau, \end{cases}$$

Model architecture. Similarly to [46], our approach leverages an encoder-only Transformer model, which has demonstrated strong performance compared to other models we tried, namely, logistic regression, fully connected neural nets (FCN) and convolutional networks. Fully connected networks of the same size as our model do achieve some good accuracies after some period of training as shown in table 8 in the appendix, but we find that they are very slow at learning and rarely find the best solution that yields 100% accuracy.

The model processes an input sequence of vectors through an embedding layer, followed by four layers consisting of *self-attention* and *feed-forward networks*. Each vector in the sequence has an embedding dimension of $d_{\text{emb}} = 1024$.

In the self-attention mechanism, the model employs multi-head attention mechanism with $h = 4$ heads, where each head operates on a subspace of dimension $d_{\text{head}} = \frac{d_{\text{emb}}}{h} = 256$. The input vectors are first projected into these lower-dimensional subspaces, processed independently by each head, and then recombined to restore the original embedding dimension.

The feed-forward network (FFN) in each block consists of two linear transformations with a *GELU* [27] non-linearity in between. It first expands the dimension to $4 \times d_{\text{emb}} = 4096$ using a fully connected layer, applies the activation function, and then projects the vectors back to the original embedding dimension d_{emb}.

After processing through these layers, the final sequence representation is obtained via *max-pooling* over the sequence length. The resulting pooled vector is then linearly projected into a scalar, which serves as the model's *logit* and is used in the loss function for optimization. We use Adam optimizer [30] with warmup ≈ 1000 steps and set the learning rate to $lr = 10^{-5}$ and weight decay to $\omega = 10^{-3}$. We use a binary cross-entropy loss function to optimize the model during training which is basically maximizing the likelihood of the training batches. To evaluate the model's performance, we measure accuracy and precision on a separate balanced test set.

Data Representation. This is a crucial factor in achieving our distinguishing results. In fact, given that the input is a standard form matrix over a finite field, multiple encoding strategies are possible, including a flat token stream, 2-D patches (table 6), and column-sequence encodings, as explained in appendix C. Among these, the row-sequence strategy is the most effective: To reflect that $\mathbf{G}$ forms a row basis of a linear code, we treat the matrix as a sequence of row vectors. Each row is first cast into a numeric vector by lifting the finite field entries to $\mathbb{R}$ using the encoding guidelines described below, then linearly projected into the Transformer's embedding space. We add an absolute positional encoding on the sequence level.

When the base field is not $\mathbb{F}_2$, we encode the field elements differently based on the value of q. If q is prime, we use angular embedding as in [46], which doubles the dimension of the rows. Otherwise, if q is a prime power, we first represent the elements as vectors of polynomial coefficients, then apply the appropriate encoding based on the prime base field.

Example: The field $\mathbb{F}_9$ can be constructed as an extension of the base field $\mathbb{F}_3$ using the irreducible polynomial $x^2 + 1$. Let z be one of its roots. $\mathbb{F}_9$ elements are expressed as $a + bz$ with $a, b \in \mathbb{F}_3$. Therefore, we represent these elements as vectors (a, b). Now to encode $\mathbb{F}_3$ elements, we use angular embedding, resulting in a 4 dimensional vector $(\cos(2\pi\frac{a}{3}), \sin(2\pi\frac{a}{3}), \cos(2\pi i\frac{b}{3}), \sin(2\pi i\frac{b}{3})) \in \mathbb{R}^4$.

4.2 Datasets

We consider Goppa and Alternant codes with a fixed code length n. For a given set of parameters — extension degree m and degree $t \in \mathbb{N}$ — we generate a dataset $\mathcal{D}_{\mathscr{F}}$ uniformly from the family $\mathscr{F}$ of codes, retaining only the codes of rank $k = n - mt$. Each generator matrix is computed in standard form. Additionally, we generate a dataset $\mathcal{D}_{\mathscr{R}}$ by uniformly sampling random linear codes with the same parameters and size, following the same procedure. We define $\mathcal{D} = \mathcal{D}_{\mathscr{F}} \cup \mathcal{D}_{\mathscr{R}}$, and use the notation $\mathcal{D}_{q,n,m,t}$ to explicitly specify the parameters when needed. It is important to note that $\mathscr{F} \subset \mathscr{R}$, meaning that, when generating the dataset for random linear codes, there is a non-zero probability that some samples might belong to $\mathscr{F}$ codes (e.g., Goppa or alternant codes). However, this probability is negligible due to the structure of the family $\mathscr{F}$ and the comparatively vast size of $\mathscr{R}$. As a result, its impact on the dataset is statistically insignificant for our analysis.

5 Experiments and Results

In this part, we present the experimental results of the `DeepDistinguisher` on alternant/Goppa codes (introduced in section 3.3) and MDPC/QC-MDPC codes (section 3.4). In the former case, we follow the methodology introduced in [43] to derive the parameters q, t, m and n (the code dimension is computed as $k = n - mt$). The approach is as follows:

- First, we fix the field size q and the extension degree m. We set the length as $n = q^m$ (full support) and find the largest t for which the code can be distinguished.
- Once such t is identified, we fix its value as well as the corresponding q and m. We then search for the smallest value of n that is still distinguishable.

Our experimental results for `DeepDistinguisher` are presented in two parts. In section 5.1, we analyze a specific set of parameters introduced in [13,43]. The code considered is relatively small (length at most 64). However, this allows explicit comparison of different distinguishers on a common benchmark. In section 5.2, we present more extensive results; pushing the practical experiments to code of length up to 1024. We conclude this part by providing experimental results for the codes underlying `BIKE`; demonstrating the flexibility of the `DeepDistinguisher` distinguisher (section 5.3).

5.1 Comparing Distinguishers for Goppa on a Small Benchmark

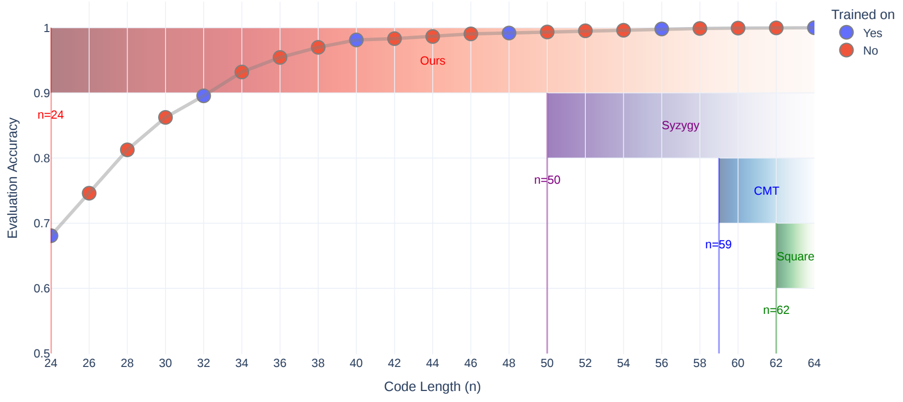

Fig. 1: Model accuracy as a function of code length. The model is trained on Binary Goppa Codes with extension degree $m = 6$ and irreducible polynomials of degree $t = 3$ as in [13] and [43]. The line and scatter points indicate the evaluation accuracy of our model on each tested value of code length. The scatter color indicates the range where the model was trained ($n = 24 + 8k$ for $k = 0, 1, ..$) showing that the model generalizes well to unseen input shapes. For attacks from the literature, we show the smallest code length reported in their respective papers, as no accuracy measures were provided.

In [13,43], the authors presented experimental results for their distinguishers on a Binary Goppa code with $q = 2, m = 6$ and $t = 3$. The maximal length is $n = 64$, the FGOPT distinguisher will be able to distinguish up to $n_{\text{square}} = 62$. The CMT distinguisher [13] reported $n_{\text{CMT}} = 59$ and [43] brings down to $n_{\text{syzygy}} = 50$.

Below this length, the conditions for distinguishability from [43] are not verified anymore.

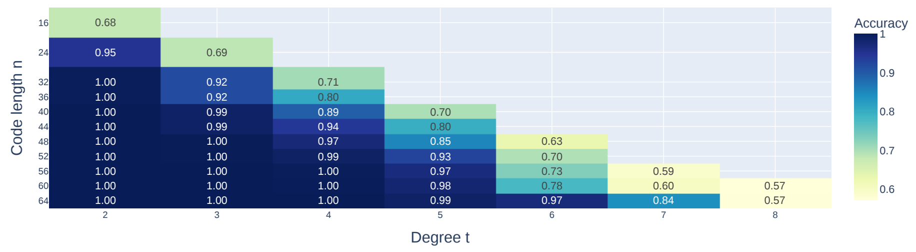

Fig. 2: Heatmap of the classification accuracy on test sets for $q = 2, m = 6$ as a function of code length n and degree parameter t. In this experiment, one model is trained per value of t only. Meaning the model is trained on codes with varying lengths n, allowing the model to interpolate well to unseen code lengths.

As highlighted in Figures 1 and 2, our distinguisher works for any code length tested, with nearly 100% accuracy for most values of $n \geq 40$. We set a lower bound on the values of n such that the rate is no less than 0.20, which, for instance, corresponds to $n > 22$ when $t = 3$. The accuracy tends to drop for very small code rates.

In this case of $q = 2, m = 6$, our distinguisher works for all values of $t \in [\![2 : 9]\!]$, and achieves perfect accuracy when $t \leq 6$. Figure 2 shows a heatmap of our model's accuracy across different values of n and t. This visualization provides a comprehensive overview of how accuracy varies with these parameters, serving as a benchmark for further investigations and comparisons with other approaches and future works.

Inference Complexity. Since we are using a standard model size throughout our work, we can give an estimate of the time complexity of our distinguisher. In fact, the complexity of this distinguisher is determined by its inference time, which corresponds to performing a forward pass through the trained model times the number of calls to the model which is usually once. This cost is proportional to the model size (with at most $\leq 50M$ parameters) and scales polynomially with the input parameters k, n. In practice, calling our model takes about 10 milliseconds (ms) on CPU (or $100ms$ in one CPU thread) and less than $\approx 1ms$ on a GPU. While training requires several hours, it is a one-time, offline process whose cost will be amortized.

5.2 Distinguishing Goppa and Alternant Codes

Goppa codes demonstrate distinguishability across a range of parameters with some specific configurations that achieve perfect accuracy. Binary codes of length $n = 128$ and extension degree $m = 7$ can be distinguished up to polynomial degree $t = 8$, with 100% accuracy for $t \leq 4$ as shown in table 2. For larger codes such as $n = 512, m = 9$, the model - of the same size as the 128-model -

distinguishes codes up to $t = 4$. In general, we observe that the accuracy of the model is lower as the degree of the Goppa polynomial t and the extension degree m increase. Experiments show that alternant codes are harder to distinguish from random codes, achieving accuracy better than random only when $t \leq 3$, for codes of length $n = 64$ and extension degree $m = 6$, as demonstrated in Table 1.

Table 1: Distinguishing accuracy for Goppa/alternant codes with $n = 64$. In this experiment, for each parameter tuple (q, n, m, t), the model is trained on a dataset $\mathcal{D}_{q,n,m,t}$ of total size ≤ 40 million samples and evaluated on 10 k unseen samples. We use a batch size $b = 512$; the number of training steps (listed below) lets the reader compute the effective training-data budget ($\min\{40m, \text{steps} \times \text{batch size}\}$) and the number of epochs ($\text{steps} \times \text{batch size}\ /40m$).

Code	(q,m)	**Degree** (t)	**Rate** (R)	**Accuracy (%)**	**Training Steps**
Goppa	$(2,6)$	2	0.81	99.12	1K
		3	0.72	98.88	8.5K
		4	0.63	98.52	22K
		5	0.53	98.24	48.5K
		6	0.44	96.68	243K
		7	0.34	84.60	848.5K
		8	0.25	57.42	90K
		9	0.16	75.92	296.5K
	$(3,4)$	2	0.88	98.25	9.8K
		3	0.81	98.02	82K
		4	0.75	90.34	154.2K
		5	0.69	54.71	113.8K
		6	0.63	52.52	44.8K
Alternant	$(2,6)$	2	0.81	57.82	15.6K
		3	0.72	53.06	14.4K
		4	0.63	51.80	18.8K

Goppa Codes with $q = 3, m = 4, n = 64$. When considering *ternary* Goppa codes with $m = 4$, we observe that the distinguishing task is more challenging compared to the binary case. Nevertheless, our distinguisher remains effective up to degree $t = 6$ (corresponding to a code rate of $R = 0.63$) as shown in Table 1. As the degree increases, accuracy drops considerably. For $t = 4$, the accuracy decreases to 90.34%, and for $t = 5$, it drops sharply to 54.71%, indicating that distinguishing becomes significantly harder.

Goppa Codes with $q = 2, m = 7$. We train the distinguisher on a 12M dataset of binary codes of extension degree $m = 7$ while varying the code lengths and degrees t. But first, we train on maximal code length $n = 128$ to figure out the highest distinguishable value of polynomial degree t. As illustrated in table 2, the model perfectly distinguishes codes up to $t = 4$ which corresponds to a code rate of $R = 0.78$; a rate that is not square-distinguishable due to the condition in

Equation (1). The accuracy starts to drop beyond that value of t but still does better than random. The highest degree t we can distinguish is $t = 8$ with a rate of $R = 0.56$ but only with a accuracy 0.52%. This rate is beyond the CMT [13] distinguishable range ($R \geq 2/3$). More details are provided in table 2.

Table 2: Distinguishing Results for Binary Goppa Codes with $n = 128$, $m = 7$. Balanced dataset $\mathcal{D}[q, n, m, t]$ of total size 40 million samples and evaluated on $10k$ unseen samples. We also show the time complexity of the classical attacks [13] $\mathbb{C}_{CMT}^{\text{sparse}} = 3\left(\binom{tm}{2} - k + 1\right)\binom{\binom{tm}{2}+d_{\text{reg}}-1}{d_{\text{reg}}}^2$. Our attack's cost is the inference cost which is less than 100 milliseconds on one CPU core for most experiments.

Goppa Degree (t)	**Rate (R)**	**Accuracy (%)**	**Training Steps**	$\mathbb{C}_{CMT}$
2	0.89	98.14	2K	-
3	0.84	99.48	91K	2^{24}
4	0.78	98.88	36K	2^{41}
5	0.73	64.52	579K	2^{65}
6	0.67	57.00	115K	2^{97}
7	0.62	54.42	411K	✗ 2^{139}
8	0.56	52.38	20K	✗ 2^{193}
9–17	0.51–0.07	≤ 51	600K	✗ $> 2^{264}$

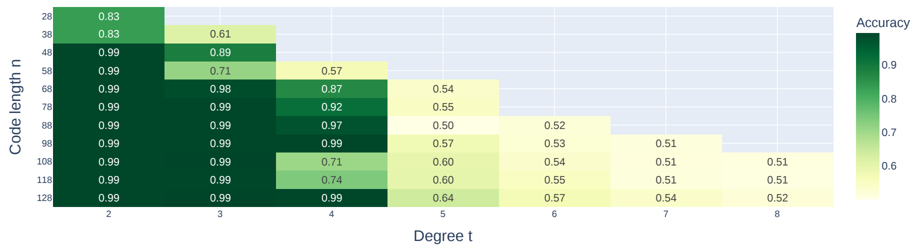

Fig. 3: Heatmap of model accuracy for classifying Goppa codes vs random codes $q = 2, m = 7$ as a function of code length n and degree parameter t.

Next, we systematically vary the code length for each degree t to identify the point at which our distinguisher fails. Figure 3 presents a heatmap of accuracies for different pairs (n, t), illustrating the performance across various code lengths and degrees. These results serve as complementary benchmarks to the $m = 6$ case, where performance appears to be saturated, providing additional insights into our distinguisher's behavior. Notably, no public implementations of classical attacks are available for direct comparison, making our results a standalone reference for this setting.

Goppa Codes $n = 256$. To test the limits of our model on larger codes, we generate datasets of $8M$ samples ($4M$ for each class) for codes of length 256. We train the model of the same size on these datasets and report the accuracies obtained in Table 3. We notice the performance degrading fast with the degree t and only do better than random for $t \leq 5$. Further efforts and resources in terms of dataset generation, model size, and training compute are needed to figure out the scaling laws of our approach.

Table 3: Classification Accuracy (%) on balanced $10k$ eval datasets of binary irreducible Goppa codes of length $n = 256$. The model used is the same throughout the paper: a tokenizer-free encoder only transformer with 4 layers and $d = 1024$ embedding dimension.

Degree (t)	2	3	4	5	6
$n = 256, m = 8$	98.06	98.38	60.36	54.74	51.66

A notable pattern during training as shown in Figure 4, is that the loss often almost stagnates for an extended period without the gradients vanishing before abruptly decreasing at a specific training step, denoted as $T_{q,m,t}$. This drop in loss tends to consistently occur much later for larger values of t, though the exact nature of the dependency between t and $T_{q,m,t}$ remains unclear. This raises a question about the applicability of gradient-based optimization on such tasks.

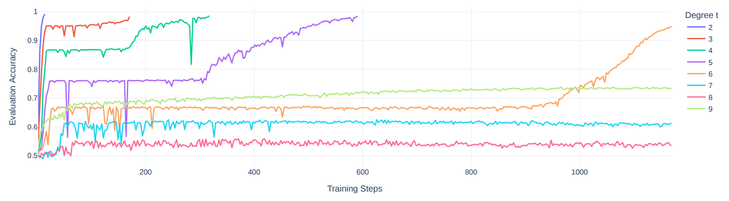

Fig. 4: Evolution of evaluation accuracy during training of the classifier over Goppa codes vs random codes with parameters $n = 64, q = 2, m = 6$. The color represents different values of the polynomial degree t.

Larger codes with Code Puncturing. We applied the strategy discussed in Appendix B to evaluate the model trained on binary Goppa codes with $m = 7, n = 128$ on codes of parameters $n = 1024, m = 10, t = 2$ using algorithm 2 with 1000 trials. This experiment yields a 70% accuracy suggesting that there are probably unknown relationships between families of binary Goppa codes over different field extensions.

5.3 Distinguishing MDPC and QC-MDPC Codes

We adopt the same framework outlined in `BIKE` [2]. Specifically, we take $\ell = 2, n = \ell r$, implying that QC-MDPC code has rate $\frac{1}{2}$ and the corresponding parity-check matrix is composed of two circulant blocks. We train the model on codes of length $n = 158$ and vary the row weight w. Taking a block size $r = 79$ prime and odd values of $w/2$ ensures that the circulants are invertible in $\mathbb{F}_q$, which explains the values of w considered in Table 4. This table shows that we can distinguish MDPC codes up to $w = 14$ while for QC-MDPC, we could only distinguish codes with row weight $w = 6$.

This outcome is somewhat surprising, as one might expect the additional structure introduced in the Quasi-Cyclic case to make classification easier rather than harder. However, the circulant structure seems to introduce constraints that makes it more challenging for the model to extract distinguishing features. An avenue of improvement is to elaborate an effective representation of this structure in a way that helps the learning of the model.

Table 4: Maximum evaluation accuracy on Moderate-Density Parity-Check (MDPC) codes versus Quasi-Cyclic MDPC codes on the distinguishing task. The number of training steps needed to achieve these accuracies is also reported.

Code length & dim (n, r)	Code	Row weight (w)	Eval Accuracy	Train. Steps
158, 79	MDPC	10	97.14	445K
		11	74.36	265K
		12	65.39	200K
		13	58.28	145K
		14	54.90	220K
		16	51.73	335K
		18	51.05	220K
	QC-MDPC	6	98.02	78K
		10	51.31	738K
		14	51.36	1.08M
		18	51.21	905K

6 Hidden Goppa Code Problem

We introduce a new problem related to Goppa codes stronger than distinguishing but weaker than the key-recovery problem.

*Problem 4 (Hidden Goppa Code (*HGC*) problem). Given a parameter $\zeta > 0$, and matrix $\widetilde{\mathbf{G}} \in \mathcal{M}_{k\times n}(\mathbb{F}_q \cup \{*\})$ with at most ζ placeholder symbols $*$, the* HGC *problem asks to find – if it exists – a completion $\widehat{\mathbf{G}} \in \mathcal{M}_{k\times n}(\mathbb{F}_q)$ of $\widetilde{\mathbf{G}}$ (i.e. obtained by replacing all placeholder symbols $*$ by field elements) such that $\widehat{\mathbf{G}}$ is a valid generator matrix for a Goppa code $\mathcal{G}(\boldsymbol{\alpha}, g)$ with $\boldsymbol{\alpha} \in \mathbb{F}_{q^m}^n$ a support and $g(x) \in \mathbb{F}_{q^m}[x]$ a Goppa polynomial of degree t.*

This problem is trivial in the case of random linear codes, since any solution makes a valid code. However, for Goppa codes, the structure imposed by the algebraic properties of the code constrains the space of possible solutions. This structure is defined by the Goppa polynomial and the support set, which are not directly visible in the generator matrix. Therefore, any solution to the HGC problem must implicitly respect these hidden parameters, making it a non-trivial task.

It is known that revealing only $tm + 1$ Goppa support points suffices to recover the McEliece secret key in polynomial time [31]. Thus, although HGC does not explicitly output any secret-key data, extracting a trained model's implicit knowledge—via interpretability or inversion techniques—could yield exactly the partial hints that classical algebraic attacks [22,21] lift to a complete key-recovery breach.

`DeepRecover`: Using the same transformer architecture as for the `DeepDistinguisher` with several hidden elements ranging between $\zeta = 1$ and $\zeta = 80$ out of $mt(n-mt)$ entries (e.g. 624 for $m = 6, t = 2, n = 64$), we were able to successfully train the model on this task achieving a component-wise accuracy as high as 80% for binary [64-52]-Goppa codes over extension degree $m = 6$ as shown in table 5.

We find that larger values of ζ accelerate the model training but converge to a less optimal solution. Further investigations are required to understand better the limits of the feasibility of this problem. It is evident that there is a theoretical upper bound of ζ beyond which the number of possible solutions explodes and we think that our model works partially because, for the values we chose of ζ, the solution is either unique or there are not many solutions, allowing the model to recover the solution that we used to generate the given sample (generate a Goppa code, hide some entries, then ask the model to recover that exact solution instead of recovering any valid solution). It's worth noting that thanks to our `DeepDistinguisher`, we could also train the `DeepRecover` model to recover any valid solution since we can test in a gradient-friendly way whether a matrix is Goppa or not with high accuracy.

Degree (t)	**Best Accuracy**
2	0.80
3	0.76
4	0.64
5	0.58
6	0.50

Table 5: Best component-wise accuracy for each polynomial degree t after training on Goppa codes with parameters $q = 2, n = 64, m = 6$. An accuracy of 50% is as good as random guessing.

7 Conclusion

`DeepDistinguisher` achieves near-perfect classification on small parameters, however, its decisions cannot be explained by any linear function of the generator-matrix entries—indeed, training a logistic-regression (which would be able to find any affine hyperplane separating Goppa codes from non-Goppa codes) baseline recovers only the total-Hamming-weight test we described in appendix A, which fails to approach 100 % accuracy and degrades rapidly as parameters grow.

Instead, our transformer must be exploiting non-linear correlations among the row-space basis vectors. In the small parameter regime, the algebraic structure of a Goppa generator matrix induces strong, low enough order correlations among its rows that our transformer's attention layers can easily pick up.
As the code rate $k/n = 1 - mt/n$ decreases by increasing t or m, the algebraic degree of the true invariants rises—forcing the model to learn ever more complex multilinear monomials—yet our fixed-size network and finite training set lack the capacity and coverage to generalize over that explosion of possibilities. Consequently, beyond a certain threshold in mt, `DeepDistinguisher`'s accuracy collapses to chance, mirroring the known intractability of distinguishing large-parameter Goppa codes from random codes.

Future explainability work should apply attention-map visualization and gradient-based saliency on the embeddings of Goppa code matrices to pinpoint exactly which combinations of row-interaction features carry the distinguishing signal.

This work is a first step in applying machine learning to code-based cryptography, opening up new possibilities for research. Future work could focus on improving these models, trying to distill a classical approach or algorithm that our model may be approximating, and figuring out the recurring behavior of the gradient descent when training on mathematical problems.

References

1. Albrecht, M.R., Bernstein, D.J., Chou, T., Cid, C., Gilcher, J., Lange, T., Maram, V., Maurich, I.V., Misoczki, R., Niederhagen, R., Paterson, K.G., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., Szefer, J., Tjhai, C.J., Tomlinson, M., Wang, W.: Classic McEliece: conservative code-based cryptography (Oct 2022), `https://inria.hal.science/hal-04288769`, round 4 submission to the NIST call for postquantum cryptographic primitives
2. Aragon, N., Barreto, P., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Ghosh, S., Gueron, S., Güneysu, T., et al.: Bike: bit flipping key encapsulation (2022)
3. Ba, J.L., Kiros, J., Hinton, G.E.: Layer normalization. arXiv preprint arXiv:1607.06450 (2016)
4. Bardet, M., Faugère, J.C., Salvy, B., Yang, B.Y.: Asymptotic behaviour of the degree of regularity of semi-regular polynomial systems. In: The Effective Methods in Algebraic Geometry Conference – MEGA 2005. pp. 1–14 (2005)
5. Bardet, M., Mora, R., Tillich, J.P.: Polynomial time key-recovery attack on high rate random alternant codes. IEEE Transactions on Information Theory (2023)
6. Berlekamp, E., McEliece, R., van Tilborg, H.: On the inherent intractability of certain coding problems. IEEE Transactions on Information Theory **24**(3), 384–386 (May 1978)
7. Bernstein, D.J.: Grover vs. Mceliece. In: Sendrier, N. (ed.) Post-Quantum Cryptography, Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6061,

pp. 73–80. Springer (2010). https://doi.org/10.1007/978-3-642-12929-2_6, http://dx.doi.org/10.1007/978-3-642-12929-2_6

8. Bernstein, D.J., Buchmann, J., Dahmen, E. (eds.): Post-quantum cryptography. Mathematics and Statistics Springer-11649; ZDB-2-SMA, Springer Berlin Heidelberg, Berlin, Heidelberg (2009), http://opac.inria.fr/record=b1128738
9. Buchberger, B.: Bruno Buchberger's PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. Journal of Symbolic Computation **41**(3-4), 475–511 (2006)
10. Buchberger, B., Collins, G.E., Loos, R.G.K., Albrecht, R.: Computer algebra symbolic and algebraic computation. SIGSAM Bull. **16**(4), 5–5 (1982)
11. Chen, L., Jordan, S., Liu, Y.K., Moody, D., Peralta, R., Perlner, R., Smith-Tone, D.: Report on post-quantum cryptography. Reasearch report NISTIR 8105, NIST (2003), http://csrc.nist.gov/publications/drafts/nistir-8105/nistir_8105_draft.pdf
12. Courtois, N.T., Finiasz, M., Sendrier, N.: How to achieve a mceliece-based digital signature scheme. In: Boyd, C. (ed.) Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2248, pp. 157–174. Springer (2001). https://doi.org/10.1007/3-540-45682-1_10, https://doi.org/10.1007/3-540-45682-1_10
13. Couvreur, A., Mora, R., Tillich, J.P.: A new approach based on quadratic forms to attack the mceliece cryptosystem. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 3–38. Springer (2023)
14. Couvreur, A., Otmani, A., Tillich, J.P.: Polynomial time attack on wild mceliece over quadratic extensions. IEEE Transactions on Information Theory **63**(1), 404–427 (2016)
15. Cox, D.A., Little, J.B., O'Shea, D.: Ideals, Varieties and Algorithms. Springer Verlag (2005)
16. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems **2**(4), 303–314 (1989)
17. Dallot, L.: Sécurité de protocoles cryptographiques fondés sur les codes correcteurs d'erreurs. (Security of cryptographic protocols based on error correcting codes). Ph.D. thesis, University of Caen Normandy, France (2010), https://tel.archives-ouvertes.fr/tel-01102440
18. Debris-Alazard, T.: Code-based cryptography: Lecture notes. CoRR **abs/2304.03541** (2023). https://doi.org/10.48550/ARXIV.2304.03541, https://doi.org/10.48550/arXiv.2304.03541
19. Faugère, J., Din, M.S.E., Spaenlehauer, P.: On the complexity of the generalized minrank problem. J. Symb. Comput. **55**, 30–58 (2013). https://doi.org/10.1016/J.JSC.2013.03.004, https://doi.org/10.1016/j.jsc.2013.03.004
20. Faugere, J.C., Gauthier-Umana, V., Otmani, A., Perret, L., Tillich, J.P.: A distinguisher for high-rate mceliece cryptosystems. IEEE Transactions on Information Theory **59**(10), 6830–6844 (2013)
21. Faugère, J., Otmani, A., Perret, L., de Portzamparc, F., Tillich, J.: Structural cryptanalysis of mceliece schemes with compact keys. Des. Codes Cryptogr. **79**(1), 87–112 (2016). https://doi.org/10.1007/S10623-015-0036-Z, https://doi.org/10.1007/s10623-015-0036-z
22. Faugère, J., Otmani, A., Perret, L., Tillich, J.: Algebraic cryptanalysis of mceliece variants with compact keys. In: Gilbert, H. (ed.) Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and

Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6110, pp. 279–298. Springer (2010). https://doi.org/10.1007/978-3-642-13190-5_14, https://doi.org/10.1007/978-3-642-13190-5_14
23. Gohr, A.: Improving attacks on round-reduced speck32/64 using deep learning. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019. pp. 150–179. Springer International Publishing, Cham (2019)
24. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge, MA (2016)
25. Goppa, V.D.: A new class of linear correcting codes. Problemy Peredachi Informatsii **6**(3), 24–30 (1970)
26. Gryak, J., Haralick, R.M., Kahrobaei, D.: Solving the conjugacy decision problem via machine learning. Exp. Math. **29**(1), 66–78 (2020). https://doi.org/10.1080/10586458.2018.1434704, https://doi.org/10.1080/10586458.2018.1434704
27. Hendrycks, D., Gimpel, K.: Gaussian error linear units (gelus). arXiv preprint arXiv:1606.08415 (2016)
28. Hornik, K.: Approximation capabilities of multilayer feedforward networks. Neural Networks **4**(2), 251–257 (1991)
29. Kim, B.D., Vasudevan, V.A., D'Oliveira, R.G., Cohen, A., Stahlbuhk, T., Médard, M.: Cryptanalysis via machine learning based information theoretic metrics. arXiv preprint arXiv:2501.15076 (2025)
30. Kingma, D.P.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
31. Kirshanova, E., May, A.: Breaking goppa-based mceliece with hints. Information and Computation **293**, 105045 (2023)
32. Li, C., Wenger, E., Allen-Zhu, Z., Charton, F., Lauter, K.: SALSA VERDE: a machine learning attack on Learning With Errors with sparse small secrets. In: Proc. of NeurIPS (2023)
33. Li, C.Y., Sotáková, J., Wenger, E., Malhou, M., Garcelon, E., Charton, F., Lauter, K.: Salsa Picante: A Machine Learning Attack on LWE with Binary Secrets. In: Proc. of ACM CCS (2023)
34. MacWilliams, F.J., Sloane, N.J.A.: The theory of error-correcting codes (1977), https://api.semanticscholar.org/CorpusID:118260868
35. McEliece, R.J.: A public-key cryptosystem based on algebraic. Coding Thv **4244**, 114–116 (1978)
36. Miller, E., Sturmfels, B.: Combinatorial Commutative Algebra, Graduate Texts in Mathematics, vol. 227. Springer-Verlag, New York (2005)
37. Misoczki, R., Tillich, J.P., Sendrier, N., Barreto, P.S.: MDPC-McEliece: New McEliece variants from moderate density parity-check codes. In: 2013 IEEE international symposium on information theory. pp. 2069–2073. IEEE (2013)
38. National Institute of Standards and Technology: Module-Lattice-Based Digital Signature Standard. Federal Information Processing Standards Publication 204, U.S. Department of Commerce (Aug 2024), https://csrc.nist.gov/pubs/fips/204/final
39. National Institute of Standards and Technology: Module-Lattice-Based Key-Encapsulation Mechanism Standard. Federal Information Processing Standards Publication 203, U.S. Department of Commerce (Aug 2024), https://csrc.nist.gov/pubs/fips/203/final
40. National Institute of Standards and Technology: Stateless Hash-Based Digital Signature Standard. Federal Information Processing Standards Publication 205,

U.S. Department of Commerce (Aug 2024), https://csrc.nist.gov/pubs/fips/205/final
41. National Institute of Standards and Technology (NIST): Post-Quantum Cryptography: Round 4 Submissions. https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions (2024), accessed: 2024-12-31
42. Overbeck, R., Sendrier, N.: Code-based cryptography, pp. 95–145. Springer Berlin Heidelberg, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-540-88702-7_4, http://dx.doi.org/10.1007/978-3-540-88702-7_4
43. Randriambololona, H.: The syzygy distinguisher. In: Fehr, S., Fouque, P.A. (eds.) Advances in Cryptology – EUROCRYPT 2025. pp. 324–354. Springer Nature Switzerland, Cham (2025)
44. Reed, I.S., Solomon, G.: Polynomial codes over certain finite fields. Journal of the society for industrial and applied mathematics **8**(2), 300–304 (1960)
45. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 1–40 (2009)
46. Stevens, S., Wenger, E., Li, C., Nolte, N., Saxena, E., Charton, F., Lauter, K.: Salsa fresca: Angular embeddings and pre-training for ml attacks on learning with errors. arXiv preprint arXiv:2402.01082 (2024)
47. Torres, R.C., Sendrier, N.: Analysis of information set decoding for a sub-linear error weight. In: Takagi, T. (ed.) Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9606, pp. 144–161. Springer (2016). https://doi.org/10.1007/978-3-319-29360-8_10, http://dx.doi.org/10.1007/978-3-319-29360-8_10
48. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems (NIPS). pp. 5998–6008 (2017)
49. Wenger, E., Chen, M., Charton, F., Lauter, K.E.: Salsa: Attacking lattice cryptography with transformers. Proc. of NeurIPS (2022)
50. Wenger, E., Saxena, E., Malhou, M., Thieu, E., Lauter, K.: Benchmarking attacks on learning with errors. Cryptology ePrint Archive, Paper 2024/1229 (2024), https://eprint.iacr.org/2024/1229

Appendices

A A Simple Statistical Hamming Weight Distinguisher

In this section, we present a simple statistical distinguisher based on an experimental observation about the distribution of generator matrices of Goppa codes. Computing the total Hamming weight of these matrices allows us to distinguish Goppa codes from *uniform* random codes, although with weaker performance compared to other methods.

We now fix the parameters under consideration, namely the code length n, extension degree m, Goppa degree t, and the dimension $k = n - mt$, and we consider random codes with the same dimension. Given a generator matrix $\mathbf{G} \in \mathcal{M}_{k \times n}(\mathbb{F}_q)$ from a code family $\mathscr{F} \in \{\mathscr{G}, \mathscr{R}\}$ (for Goppa versus random), we define its total Hamming weight as

$$w_H(\mathbf{G}) = \sum_{i=1}^{k} w_H(\mathbf{g}_i),$$

where $\mathbf{g}_i$ is the i-th row of $\mathbf{G}$. The variable $w_H(\mathbf{G})|_{\mathscr{F}}$ is a random variable that takes discrete values, with the distribution denoted as $p_{\mathscr{F}}$. To check for any differences in the behavior of $w_H(\mathbf{G})|_{\mathscr{F}}$ between the two distributions, we empirically estimate the total variation distance between $f_{\mathscr{G}}$ and $f_{\mathscr{R}}$ using $1M$ samples:

$$D_{TV}(\hat{f}_{\mathscr{G}}, \hat{f}_{\mathscr{R}}) = \frac{1}{2} \sum_{x \in \mathcal{X}} |\hat{f}_{\mathscr{G}}(x) - \hat{f}_{\mathscr{R}}(x)|.$$

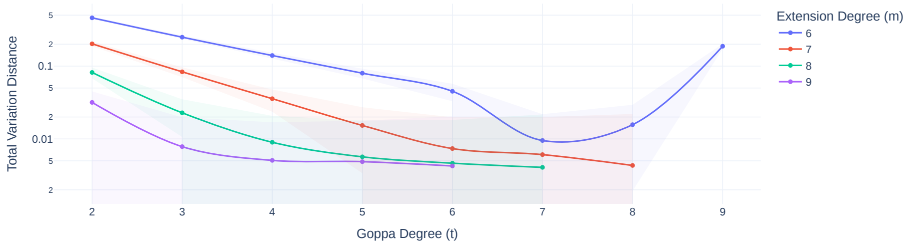

Fig. 5: Total variation distance of empirical distributions $p_{\mathscr{G}}$ (Goppa) and $p_{\mathscr{R}}$ (Random) of the total Hamming weight metric for binary codes of length $n = 64$ with varying extension degree m and polynomial degree t. The distance axis is in log scale showing the distance exponentially converging to 0. The plateauing might be because of the estimation error being high. 1M samples were used for probability estimation.

The empirical distributions $\hat{f}_{\mathscr{G}}$ and $\hat{f}_{\mathscr{R}}$ are shown as histogram plots in the appendix Figure 6 for codes of length $n = 64$ and extension degree $m = 6$. We can see that the distributions don't match especially for small values of t and exceptionally for $t = 9$ when $n = 64$ and $m = 6$ which is not the case for larger codes. We show this by also varying the extension degree and plotting the empirical TV distance in Figure 5 as a function of m and t. The distance is significant for small values of t but exponentially decreases.

To devise a statistical distinguisher, we can employ a hypothesis test based on the likelihood ratio. In the random case, the distribution of the total Hamming weight is known; a binomial distribution since it's a sum of independent Bernoulli variables with $p_{\mathscr{R}} = 0.5, N_{\mathscr{R}} = mt(n - mt)$ (only codes in standard form are considered). For the Goppa case, we make an assumption that the distribution is also a binomial and empirically estimate $\hat{p}_{\mathscr{G}}$ and $\hat{N}_{\mathscr{G}}$ using some training data.

Given a sample $x = w_H(\mathbf{G})$, we compute the likelihood ratio:

$$\log \Lambda(x) = \log \frac{\binom{\hat{N}_{\mathscr{G}}}{x}}{\binom{N_{\mathscr{R}}}{x}} + x \log \frac{\hat{p}_{\mathscr{G}}}{p_{\mathscr{R}}} + \log \frac{(1 - \hat{p}_{\mathscr{G}})^{\hat{N}_{\mathscr{G}} - x}}{(1 - p_{\mathscr{R}})^{N_{\mathscr{R}} - x}}$$

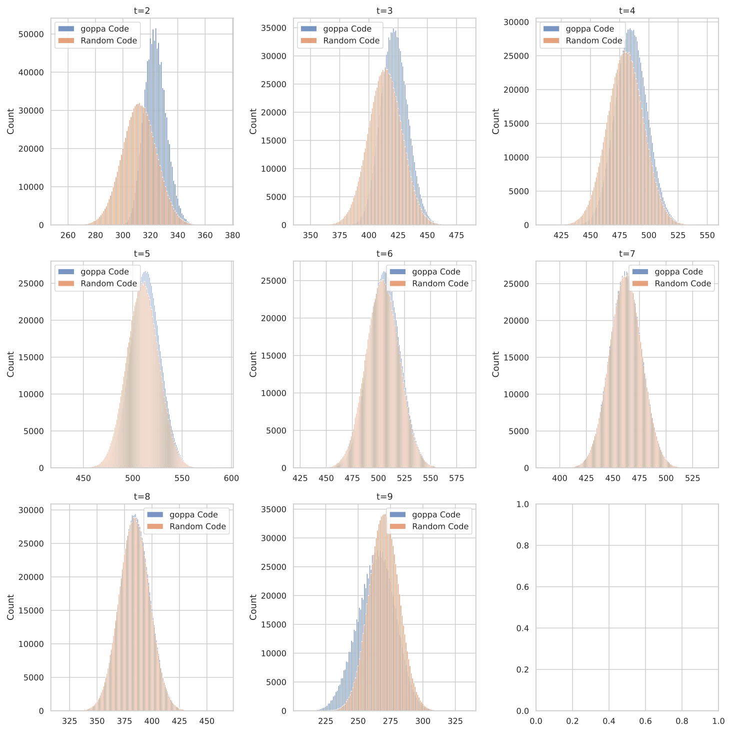

Fig. 6: Histograms of the total Hamming weight of the generator matrix of Goppa codes versus random linear codes: $w_{\mathscr{F}}(\mathbf{G}) = \sum_{i=1}^{k} w_H(g_i)$ for binary codes with parameters $n = 64, m = 6$ while varying polynomial degree t.

The distinguisher can therefore be expressed as follows:

$$\mathscr{D}(\mathbf{G}) = \mathbb{1}_{\{\log \Lambda(x) > \tau\}}(w_H(\mathbf{G}))$$

Using this simple distinguisher with $\tau = 0$, we can achieve a test accuracy of 73% on a balanced 1M dataset with the parameters of the first graph ($t = 2, n = 64, m = 6$) of Figure 6 and 62% accuracy on $t = 3$ and only 57% for $t = 4$.

It's worth noting that by changing the sampling distribution over random linear codes from uniformly random to $\mathcal{B}(p)^{k\times(n-k)}$ (independent Bernoulli entries) for p matching the experimental value for Goppa codes of the same parameters, this distinguisher will degrade.

B Out-of-distribution Evaluation: Punctured Codes

To evaluate the distinguisher on instances of larger code lengths, we puncture the code by truncating the public generator matrix to fit into the training shape and assess the model on the resulting code. Initially, this approach does not yield satisfactory results when applied just once. However, by repeatedly truncating the original matrix through subsampling of rows and columns, and evaluating

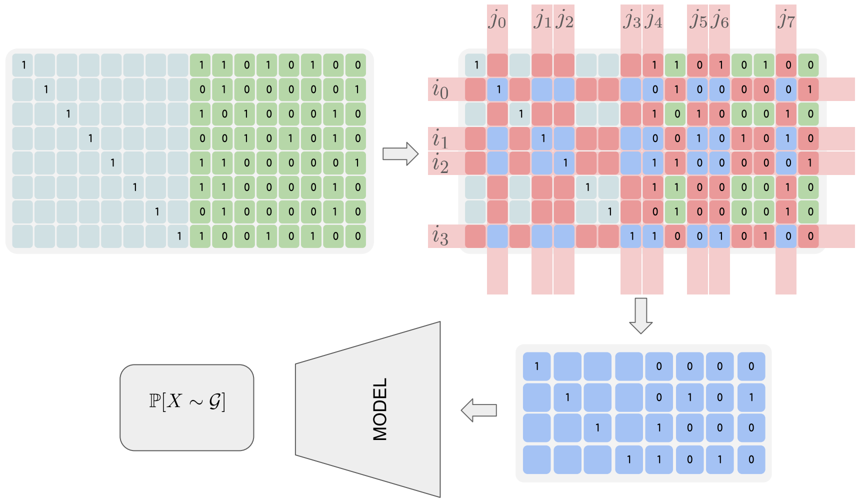

Fig. 7: To assess an $[8, 4]$-model (trained on codes of length $n = 8$ and dimension 4) on a $[16, 8]$-code, we puncture the input code by sampling $i_0 = j_0, \dots, i_3 = j_3$ and $j_4, \dots, j_7$ randomly to create a new $[8, 4]$-code in standard form.

the model on each of these subsampled matrices, we can improve performance. By aggregating the results, we determine an optimal decision threshold based on the number of positive classifications or vote counts on a validation set.

Algorithm 1 Sample Punctured SubCode

```
1: procedure SamplePuncturedSubCode(G)
2:     Input: Generator matrix $\mathbf{G} = (\mathbf{I}_{k_0} \mid \mathbf{A}) \in \mathbb{F}_q^{k_0 \times n_0}$
3:     Output: Punctured subcode matrix $\mathbf{G}[\mathcal{I}, \mathcal{J}]$
4:     Sample valid $(n, k)$ such that $k \leq k_0$ and $n - k \leq n_0 - k_0$
5:     Sample $k$ row indices $\mathcal{I}$ from $[\![0; k_0 - 1]\!]$ without replacement.
6:     Sample $n - k$ column indices $\mathcal{J}'$ from $[\![k_0; n_0 - 1]\!]$
7:     Form the set of column indices $\mathcal{J} = \mathcal{J}' \cup \mathcal{I}$
8:     return submatrix $\mathbf{G}[\mathcal{I}, \mathcal{J}]$
9: end procedure
```

More formally, given $\mathbf{G} = \begin{pmatrix}\mathbf{I}_{k_0} & \mathbf{A}\end{pmatrix} \in \mathbb{F}_q^{k_0 \times n_0}$ a standard form generator matrix of a linear code $\mathcal{C} \in \mathscr{F}$, to evaluate the model on punctured subcodes of $\mathcal{C}$, we first sample one of the training code parameters (k, n) such that $k \leq k_0$ and $n - k \leq n_0 - k_0$. Then sample k row indices $\mathcal{I}$ and $n - k$ column indices $\mathcal{J}'$ from the matrix $\mathbf{A}$ in addition to the k columns forming the identity matrix of shape k: $\mathcal{J} = \mathcal{J}' \cup \mathcal{I}$. This is important because during training, the model only sees standard form matrices so we don't expect it to generalize to unseen input in those first columns. Therefore, the identity matrix acts as a positional encoding of the sequence. Finally, we assess the model on $\mathbf{G}[\mathcal{I}, \mathcal{J}]$.

Algorithm 2 Evaluate distinguisher on larger codes using puncturing.

```
Require: Generator matrix $\mathbf{G} = (\mathbf{I}_{k_0} \mid \mathbf{A}) \in \mathbb{F}_q^{k_0 \times n_0}$, number of trials $m$
Ensure: Aggregated result of model evaluations
1: function EvaluateModel($\mathbf{G}, k_0, n_0, m$)
2:     Initialize result_sum ← 0
3:     for $i = 1$ to $m$ do
4:         $\mathbf{G}_{\text{punc}}$ ← SamplePuncturedSubCode($\mathbf{G}, k_0, n_0$)
5:         result ← EvaluateDistinguisher($\mathbf{G}_{\text{punc}}$)
6:         result_sum ← result_sum + result
7:     end for
8:     return result_sum
9: end function
```

C Possible Data Representations

Among the key questions raised during the experiments is how to best represent numerically the input matrices $\mathbf{G}$ as sequences of numerical values/vectors which is the standard input to a transformer.

Flat token stream. A naïve language-model baseline flattens the generator matrix G row by row, inserting a special end-of-row separator after each row. Every finite-field symbol is then treated as a categorical token and embedded via a randomly-initialised lookup table. Although straightforward, this representation is inefficient: the resulting sequence length is kn (or $k(n-k)$ if the identity block of a systematic form is omitted), and the attention cost grows quadratically with that length. Moreover, the model must learn the algebraic relationships between field elements entirely from data, because none of that structure is provided *a priori.*

2-D patches in order to reduce the number of tokens, one can think of fragmenting the matrix $\mathbf{G}$ into patches and assign a token id to each distinct patch. The patch size is quantified by *height* h × *width* w. A large patch size would result in a shorter sequence but with a large vocabulary since patches become more distinct from each other. Some results are reported in table 6.

A **column-sequence** representation is certainly feasible and experimented with; however, our experiments show that the row-sequence approach delivers the strongest results. The next-best option is a patch-embedding scheme with patch height $h = 1$, which essentially preserves the row-wise structure and therefore benefits from a similar inductive bias.

D Implementation details

Datasets are generated using SageMath and saved in files for training. One noticeable artifact that occurs when standardizing non standard form binary codes in SageMath is that the resulting distribution of the generator matrix entries are not uniform due to the algorithm used to swap columns. Plotting the probabilities that an entry is 1 shows that some cells are less likely to be one. This occurs regardless of the code family. One way to avoid this is to simply

Extension degree	Polynomial degree	Tokenization Patch	Eval Accuracy	Eval recall	Eval Specificity
8	2	1x4	0.711	0.771	0.649
		1x8	0.693	0.763	0.619
		4x1	0.601	0.762	0.446
		4x4	0.592	0.620	0.563
		8x1	0.585	0.710	0.458
8	3	1x4	0.529	0.725	0.334
		1x8	0.550	0.694	0.406
		4x1	0.534	0.545	0.524
		4x4	0.512	0.473	0.551
		8x1	0.529	0.640	0.417
8	4	1x4	0.530	0.323	0.732
		1x8	0.521	0.630	0.407
		4x1	0.526	0.608	0.444
		4x4	0.514	0.391	0.638
		8x1	0.516	0.610	0.425

Table 6: Performance of the `DeepDistinguisher` distinguisher on classifying Goppa code generator matrices over $\mathbb{F}_q$ (here $q = 2$) when tokenizing the inputs by 2D patches, showing evaluation accuracy, recall, and specificity for different polynomial degrees and tokenization patch sizes. The data parameters are $q = 2, m = 8, n = 128$

discard non standard codes meaning we keep only about 29% of the codes. We use a small transformer of size ≈ 50 million trainable parameters with 4 layers and embedding dimension $d = 1024$. Our implementation is using torch and is based on the public implementation of SALSA [49].

E Model Architecture Ablations

We perform a detailed comparison of four neural architectures-Fully-Connected Network (FCN), LeftRight FCN, Attention FCN, and DeepDistinguisher—on binary Goppa codes with parameters $m = 6$, $n = 64$, across polynomial degrees $t \in \{2, 3, 4, 5, 6\}$. All models are matched to roughly 23–28 million parameters to ensure a fair comparison of efficiency and performance.

`DeepDistinguisher` consistently achieves the highest evaluation accuracy and recall for every degree t, and does so with markedly fewer training steps than the competing architectures. At lower degrees ($t = 2, 3$), all models achieve high accuracy (≥ 0.965), but `DeepDistinguisher` converges in only 1000 and 24500 steps, respectively—an order of magnitude faster than Attention FCN and two orders faster than FCN variants. As t increases, the performance gap widens: for $t = 4, 5$, `DeepDistinguisher` maintains near-perfect accuracy (0.980–0.984) with moderate training times (133 000–569 500 steps), while the other models suffer substantial drops in recall and overall accuracy. Even at the highest complexity

Table 7: Fixed experimental hyperparameters for $n = 64$

Parameter	**Value**
Model	encoder-only
Dataset	Goppa code dataset ($n = 64$), 6 M samples per class
Batch size (train / val)	512 / 2000
# training samples	12 000 000
# max epochs	30 or accuracy ≥ 0.99
Eval samples per validation	20 000
Validation interval	every 1000 steps
Optimizer	AdamW with linear warmup
• Learning rate	1×10^{-5}
• Warmup steps	1000
• Weight decay	1×10^{-3}
Gradient clipping norm	5.0
Encoder architecture	4 layers, 4 heads, 1024-d embeddings
Positional timescale	40
Precision	float16

($t = 6$), `DeepDistinguisher` attains 0.961 accuracy and perfect specificity in under one million steps, whereas the alternatives fall below 0.650 accuracy.

Polynomial degree	Architecture	Eval Accuracy	Eval Recall	Eval Specificity	Train Min Loss	Steps
2	Attention FCN	0.982	0.972	0.992	0.044	11500
2	DeepDistinguisher	**0.990**	0.980	1.000	0.060	1000
2	LeftRight FCN	0.981	0.966	0.996	0.077	65000
2	FCN	0.981	0.966	0.996	0.069	123000
3	Attention FCN	0.980	0.963	0.996	0.060	423000
3	DeepDistinguisher	**0.983**	0.971	0.995	0.054	24500
3	LeftRight FCN	0.971	0.955	0.986	0.057	1109000
3	FCN	0.965	0.934	0.995	0.080	826000
4	Attention FCN	0.865	0.747	0.980	0.281	1226500
4	DeepDistinguisher	**0.980**	0.965	0.995	0.048	569500
4	LeftRight FCN	0.887	0.788	0.988	0.254	1051500
4	FCN	0.866	0.747	0.980	0.229	577500
5	Attention FCN	0.765	0.536	0.988	0.443	1039500
5	DeepDistinguisher	**0.984**	0.973	0.995	0.066	133000
5	LeftRight FCN	0.765	0.538	0.985	0.409	567500
5	FCN	0.688	0.551	0.828	0.281	1149500
6	Attention FCN	0.620	0.454	0.790	0.265	319000
6	DeepDistinguisher	**0.961**	0.922	1.000	0.112	972000
6	LeftRight FCN	0.654	0.401	0.911	0.440	669000
6	FCN	0.543	0.387	0.696	0.419	291500

Table 8: Comparison of evaluation accuracy, recall, specificity, minimum training loss, and number of training steps for each model architecture across Goppa polynomial degrees t on data of binary Goppa codes with $m = 6, n = 64$. The considered models here are Fully-Connected Neural Network (FCN), `DeepDistinguisher` which the encoder only transformer, Left Right FCN which is a model that consists of applying an FCN on the left of the input matrix and on the right, operating on both rows and columns of the input matrices. Attention FCN is a custom encoder with self-attention layer and FCN only. Models are parameterized such that they have approximately the same size of $23 - 28$ million parameters.