TÉCNICO LISBOA

Sparse Transformers for High Order Epistasis Detection

Miguel Ângelo da Silva Graça

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Doctor Aleksandar Ilic
Doctor Leonel Augusto Pires Seabra de Sousa

Examination Committee

Chairperson: Doctor Pedro Filipe Zeferino Tomás
Supervisor: Doctor Aleksandar Ilic
Member of the Committee: Doctor Rui Fuentecilla Maia Ferreira Neves

November 2022

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa

Acknowledgments

First, I would like to thank my supervisors Aleksandar Ilic, Sergio Santander-Jiménez, and Leonel Sousa for their help, availability, and patience throughout the entire development of this thesis. I would also like to thank IST, INESC-ID, FCT, and the consortium members of EuroHPC Joint Undertaking through the Grant No. 956213 (SparCity), specially Simula, for providing the resources which made this thesis possible.

I want to acknowledge my family and thank everyone for their continuous love, support, and advice, not only during the development of this thesis, but throughout my entire life.

At last, to my friends who have supported me during the development of this thesis: to Eduardo Cunha, who knows how hard the last five years were; to João Pinheiro, whose optimism always put a smile in my face; to Sérgio Carrôlo, who always motivated me to do my best; to Teresa Medeiros, who always listened to me when I had a bad day. Thank you for all the laughs and for all the moments we shared together, good and bad.

Abstract

Genome-Wide Association Studies (GWAS) aim to identify relations between Single Nucleotide Polymorphisms (SNPs) and the manifestation of certain diseases, which is an important challenge in biomedicine. However, most genetic diseases are not only explained by the effects of individual SNPs, but by the interactions between several SNPs, known as epistasis. Detecting high order epistasis is a very computationally demanding task, due to the exponential increase in evaluated combinations of SNPs. Recently, deep learning has emerged as a possible solution for genomic prediction, but the black-box nature of neural networks and lack of explainability is a drawback yet to be solved. In this dissertation, a new framework for interpreting neural networks for epistasis detection is presented. Using sparse transformers, a technique not yet employed for epistasis detection, SNPs can be assigned attention scores to quantify their relevance for predicting a phenotype. This new methodology is proposed to be tested on IPUs, a recent massively parallel processor aimed at machine learning workloads and efficient processing of sparse data. The results on simulated datasets show that the proposed framework outperforms state-of-the-art methods for explainability, identifying SNP interactions in various epistasis scenarios. Furthermore, training on IPUs provides higher performance than GPUs and TPUs, achieving reasonable speedups up to 2.79x. To conclude, the proposed framework is validated on a real breast cancer dataset, identifying second to fifth order interactions in the top 40% most relevant SNPs.

Keywords

Genome-Wide Association Studies, Epistasis Detection, Machine Learning, IPU

Resumo

Os Estudos de Associação do Genoma Completo (GWAS) procuram identificar relações entre polimorfismos de nucleótido único (SNPs) e a manifestação de certas doenças, o que constitui um importante desafio na biomedicina. Contudo, a maioria das doenças genéticas não são apenas explicadas pelos efeitos de SNPs individuais, mas também pelas interações entre vários SNPs, conhecidas como epistasia. A deteção de epistasia de ordem elevada é um desafio computacional, devido ao aumento exponencial nas combinações de SNPs avaliadas. Recentemente, a aprendizagem profunda emergiu como uma solução possível na previsão de doenças, mas a dificuldade de interpretação dos modelos (caixa preta) é uma desvantagem por resolver. Nesta dissertação, apresenta-se uma nova metodologia para a interpretação de redes neuronais aplicadas à deteção de epistasia. Usando transformadores esparsos, uma técnica nunca usada para deteção de epistasia, atribuem-se aos SNPs uma pontuação de atenção para quantificar a sua relevância na previsão de doenças. Propõe-se a testagem desta nova metodologia em IPUs, um processador paralelo massivo recente dirigido para aprendizagem automática e processamento eficiente de dados esparsos. Os resultados em dados simulados mostram que a metodologia proposta supera os métodos do estado da arte para a interpretação de redes neuronais, identificando interações entre SNPs em vários cenários de epistasia. Mais ainda, o treino em IPUs permite ganhos de desempenho até 2.79x face a GPUs e TPUs. Para concluir, a metolodogia proposta é validada em dados reais de cancro da mama, identificando interações de segunda a quinta ordem nos 40% SNPs mais relevantes.

Palavras Chave

Estudo de Associação do Genoma Completo, Deteção de Epistasia, Aprendizagem Automática, IPU

Contents

List of Figures

List of Tables

Acronyms

ACO	Ant Colony Optimization
BNNs	Bayesian Neural Networks
CNNs	Convolutional Neural Networks
DNA	Deoxyribonucleic Acid
DNNs	Deep Neural Networks
DSAs	Domain-Specific Architectures
FPGAs	Field Programmable Gate Arrays
GAs	Genetic Algorithms
GPUs	Graphics Processing Units
GWAS	Genome-Wide Association Studies
HWE	Hardy-Weinberg Equilibrium
IPUs	Intelligence Processing Units
LRP	Layerwise Relevance Propagation
MAF	Minor Allele Frequency
MDR	Multifactor Dimensionality Reduction
MLP	Multilayer Perceptron
NLP	Natural Language Processing
SAE	Stacked Autoencoders
SM	Streaming Multiprocessors
SNPs	Single Nucleotide Polymorphisms
TPUs	Tensor Processing Units

Chapter 1

Introduction

Genome-Wide Association Studies (GWAS) have provided insight into the relationships between Single Nucleotide Polymorphisms (SNPs) and complex diseases for the past 20 years. The common workflow of a GWAS follows a case-control design [1], where individuals are defined by the presence (case) or absence (control) of a certain phenotype, which is a set of observable characteristics, such as a disease. This methodology is based on the ”common disease-common variant” hypothesis [2], which suggests that common diseases have common underlying influential variants across the population. Some successes of GWAS include the discovery of the Complement H Factor gene and its influence in age-related macular degeneration (AMD) [3] and the quantification of inherited susceptibility to obesity using polygenic predictors [4].

The approach for GWAS considers that SNPs have independent effects to the phenotype and test each SNP individually for statistical relevance to the disease [5]. However, this approach considers only single SNP effects and neglects possible effects from gene-environment and gene-gene interactions. Therefore, only a small portion of the genetic variance explains the observed phenotype, with the remaining part being referred to as ”missing heritability” [6]. The problem of missing heritability is partly due to the effects that arise from interactions between two or more SNPs. This phenomenon of combinatorial effect between SNPs is known as epistasis. Epistasis detection focuses on the identification of interactions of SNPs that are responsible for a certain phenotype, such as a disease. This approach has been successfully used to explain complex diseases, such as rheumatoid arthritis [7] and Late Onset Alzheimer's Disease [8].

While initial GWAS studies focused on approximately 1000 SNPs, SNP arrays of 500 000 - 5 000 000 SNPs are now employed and are customizable [9], as a consequence of rapid improvements in DNA sequencing technologies. Therefore, evaluating combinations of interacting SNPs is a current computational challenge. For example, in a dataset with 500 000 SNPs, there are 125 possible billion combinations of two genes (second order epistasis) and 20 quadrillion combinations of three genes (third

order epistasis). However, certain diseases are only explained when combinations of three or more SNPs are considered (high order epistasis). As the search order increases, the number of possible epistatic combinations increases exponentially, resulting in an intractable problem, due to infeasible execution times for evaluation of all combinations.

Consequently, most exhaustive methods (i.e., methods that analyze all SNP combinations up to a given order) only target second, third, and, in recent implementations, fourth order interactions. Other methods to tackle this problem attempt to reduce the search space, either by removing SNPs (filtering) or by using machine learning methods, such as Bayesian networks or deep learning, to identify promising SNP combinations. In general, these approaches are less accurate when compared to exhaustive methods, since not all possible combinations are explored.

1.1 Motivation

The high computational cost of evaluating all possible SNP combinations limits exhaustive search methods to low order interactions. Furthermore, high order interactions (e.g., three or more SNPs) are likely to have crucial implications on certain diseases [10]. As such, the need arises to derive more efficient epistasis detection algorithms that take advantage of technological advances in recent computer hardware. As an example, Graphics Processing Units (GPUs) are a widely used platform for epistasis detection. GPUs are efficient in exploring data parallelism, as multiple operations can be executed in parallel. For this reason, GPUs have been proven effective for second order [11, 12], third order [13–16], and, more recently, fourth order epistasis detection [17]. Along with GPUs, Tensor Processing Units (TPUs) have also become increasingly popular as accelerators for second order and third order epistasis [13, 16].

Another tendency to overcome the limitations of exhaustive search is to use Domain-Specific Architectures (DSAs), specifically made to offer better performance and energy efficiency for certain applications. As an example, the use of Field Programmable Gate Arrays (FPGAs) has proven successful in accelerating epistasis detection for second order [18], third order [19], and, more recently, fourth order epistasis detection [20].

While exhaustive search methods for fourth order epistasis already exist, they only proved successful for datasets with hundreds [17] to a few thousands of SNPs [20], suggesting that increasing the efficiency of exhaustive search may not be a feasible approach to tackle high order interactions in datasets with hundreds of thousands of SNPs. Within the non-exhaustive methods, deep learning arises as a promising tool for epistasis detection. Studies suggest that Deep Neural Networks (DNNs) are able to capture relationships between SNPs and the observed phenotype [21] and have good performance in large datasets [22, 23]. However, the biological interpretation is often ignored, as DNNs are black box

models and extracting SNP interactions is not a trivial task. Furthermore, the size of the network is also a limiting factor. As the number of parameters increases, the obtained accuracy also increases, albeit at the cost of memory and computation capacities. Recent studies suggest that sparse networks can achieve performance equal to dense networks [24, 25], although the irregular nature of sparse data leads to problems in memory accesses and load balancing.

Recent innovations in technology have introduced the Intelligence Processing Units (IPUs) as a novel kind of massively parallel architecture, targeted for machine learning workloads. While IPUs have not yet been applied to epistasis detection, they have already been benchmarked for machine learning problems with state-of-the-art results [26–29]. Its design premise is the efficient execution of fine-grained operations across a large number of parallel threads. Unlike other platforms, IPUs adapt well to irregular computation and data accesses, which is fitting for sparse data. Therefore, it is of interest to develop sparse machine learning methods that run on IPUs and are capable of identifying SNPs that are relevant for the observed phenotypes for high order epistasis detection. Developing and implementing this new approach is the main objective of this thesis.

1.2 Objectives

This thesis falls under the scope of EU Project Sparcity, being developed at INESC-ID. One of the goals of this project is exploiting parallel processing in bioinformatics applications (as is the case of epistasis detection) in emerging heterogeneous systems for high performance and energy efficient computing. Accordingly, the work presented here explores IPUs to implement high order epistasis detection methods.

As suggested in [26–29], IPUs adapt well to sparse data and support machine learning models with state of the art results. Therefore, this thesis has the following objectives:

- Designing accurate sparse machine learning methods for high order epistasis detection that provide a better interpretation of SNP interactions and are not yet existent in the literature;
- Implementing the developed models on Graphcore IPUs;
- Investigating different alternative codifications for the representation and processing of SNP data;
- Examining potential impacts in the time-to-solution of the proposed methods across state-of-the-art hardware for machine learning: NVIDIA GPUs, Graphcore IPUs, and Google TPUs.

1.3 Contributions

This work was supported by the FCT (Fundação para a Ciência e a Tecnologia, Portugal) and EuroHPC Joint Undertaking through the Grant No. 956213 (SparCity). To bridge the gap between explainability and predictions in deep learning models, a novel methodology and processing framework based on transformers is proposed in this thesis to tackle epistasis detection. Transformers [30] are a kind of neural network that has achieved state-of-the-art results for Natural Language Processing (NLP) tasks, although it has not yet been used for epistasis detection. Using this network, SNPs need to be represented as dense vectors known as embeddings. The proposed implementation provides several options on how to create these embeddings. For interpretation, the proposed framework leverages the attention scores that the transformer assigns to each SNP during training, which are used in the post-processing step to understand the relevance of each SNP to predict a patient's phenotype. To provide scalability of the transformer for large datasets, the proposed framework also focuses on sparsity for reduced storage and efficient computing during training.

The proposed framework is scalable, configurable, and is able to find interacting SNPs under a variety of epistasis models for interactions ranging from second to fifth order. Furthermore, it is developed on Graphcore IPUs, obtaining significant performance gains when compared to the state-of-the-art platforms for machine learning (i.e., NVIDIA GPUs and Google TPUs). For validation, the proposed framework is applied to a real breast cancer dataset, identifying second, third, and fourth order interactions among the top 30% most relevant SNPs and a fifth order interaction among the top 40%, demonstrating its reliability for epistasis detection.

1.4 Outline

This dissertation is structured as follows. Chapter 2 introduces the fundamental notions behind epistasis and presents a summary of the state-of-the-art techniques, with a particular focus on machine learning methods. Chapter 3 provides a description of the proposed framework, explaining how the transformer can be applied for epistasis detection and the methodology to interpret its predictions. Chapter 4 provides the transformer's results on a wide variety of epistasis scenarios, comparing it with other state-of-the-art techniques for network interpretability. IPUs are compared to GPUs and TPUs, regarding the necessary time each platform needs to train the transformer. To conclude and validate the approach, the transformer is applied to a real breast cancer dataset. Finally, Chapter 5 presents the most relevant conclusions from this dissertation, while also providing guidelines for future work and improvements.

Chapter 2

Background and State of the Art

An important challenge in biomedicine is the assessment of the hereditary basis for diseases. The dramatic increase in available genetic data in the last 20 years has facilitated the development of Genome-Wide Association Studies (GWAS) [9]. The main goal of GWAS is to study genetic variants within multiple individuals that display different phenotypes to find correlations between genetics and diseases. In GWAS, SNPs, the most common types of mutations in the human genome, are compared between individuals affected (cases) and unaffected (controls) by the studied disease. However, only the independent effects of individual SNPs in the observed phenotype are modeled, which is insufficient to understand complex diseases [6].

Epistasis detection is an approach in GWAS that departures from searching single SNP effects and aims to find meaningful interactions between two or more SNPs to explain complex phenotypes [10]. To find interactions between SNPs, the optimal approach is to exhaustively search all SNP combinations for a given order. However, when performing exhaustive search, a large processing power is required, as the number of combinations to analyze increases exponentially with the order of interactions and the number of SNPs. As an example, in [31], the analysis of second order interactions in a 500 000 SNP dataset (which amounts to 125 billion combinations) took 19 hours using 2 quad-core Intel Xeon processors @ 2.4 GHz. Other approaches consider non-exhaustive methods to reduce the complexity of the epistasis detection. However, due to infeasible running times, high order epistasis detection is barely tackled in the existing literature, even with non-exhaustive methods.

To better understand the problem, this chapter covers the fundamental basics for genetics to explain SNPs and their relation to GWAS. A formal definition for epistasis is given, followed by a thorough overview of the state-of-the-art approaches for epistasis detection. The most relevant methods are examined, with a specific focus on machine learning and neural networks. Furthermore, the use of specific hardware to accelerate epistasis detection is studied. Finally, a discussion on the analyzed approaches is provided to give insight on the open research challenges of epistasis detection, specially

the ones tackled in the scope of this master thesis.

2.1 Fundamentals on Genetics

To understand the phenomenon of epistasis, it is important to define the basics on genetics. Genetic information is stored in a double-stranded molecule known as Deoxyribonucleic Acid (DNA). Each strand is composed of nucleotides, linked by covalent bonds. Each nucleotide is composed of a sugar called deoxyribose, a phosphate group, and one of four nitrogenous bases: cytosine [C], adenine [A], thymine [T], or guanine [G]. Strands of DNA are connected through hydrogen bonds between nitrogenous bases, in the form of the pairs A-T and C-G.

Genes are sequences of nucleotides that store genetic information regarding an individual's phenotype. The human genome is characterized by 23 pairs of chromosomes, inside of which genes occupy specific fixed positions known as *locus*. A variant of a gene in the same *locus* is named an allele. Each *locus* is defined by two alleles, one in each chromosome of the pair. Alleles can be dominant or recessive. Dominant alleles show their effect even if the individual only has one copy of the allele, while recessive alleles only manifest their effect if the individual has two copies of the allele. Each allele is inherited by one of the parents. If both alleles are of the same type, the *locus* is considered homozygous. Otherwise, it is considered heterozygous. In the case of a heterozygous *locus*, the manifested phenotype is determined by the dominant allele.

First proposed in 1996 [32], GWAS aim to find genetic variations that are associated with a certain phenotype. SNPs are a common genetic variation employed in GWAS and represent a difference in a single nucleotide at a specific position in the DNA. As an example, for a specific DNA position, where in most individuals the nucleotide adenine (A) appears, a small group of people instead have the nucleotide guanine (G). This difference in a single nucleotide results in the occurrence of two different alleles for a specific locus, of which the least common one should appear, at least, on 1% of the population to be considered a SNP. It is estimated that the human genome has around 10 million SNPs [9].

SNPs assume three possible states: *Homozygous Major* (both alleles are dominant), *Heterozygous* (the alleles are different), and *Homozygous Minor* (both alleles are recessive). Table 2.1 depicts how each SNP state is encoded, with *Homozygous Major*, *Heterozygous* and *Homozygous Minor* represented by 0, 1, and 2, respectively.

In GWAS, SNPs are considered to have independent effects on the observed phenotype. GWAS rely on case-control studies by comparing arrays of SNPs from individuals that exhibit the disease (cases) and individuals that do not exhibit the disease (control). Typically, to identify SNPs that are correlated with the observed phenotype, the allelic frequencies (frequency of an allele at a specific *locus*) between cases and controls are compared. Statistical tests are employed to find correlations between single

SNPs and the disease state.

Table 2.1: SNP Encoding

Alleles	Genotype	Symbol
A/A	Homozygous Major	0
A/a a/A	Heterozygous	1
a/a	Homozygous Minor	2

2.2 Epistasis Detection: Overview

The standard approach for GWAS only regards additive effects, as SNPs are considered to have independent effects on the observed trait. This approach ignores gene-gene interactions, which may explain in part the problem of "missing heritability" [6]. The interaction between genes to define a phenotype is referred as epistasis. First introduced by Bateson in [33], epistasis, in a biological sense, described the effect that an allele on one *locus* had at masking an allele at a different *locus*. The resulting effect is similar to dominance but at an inter-*loci* level.

A departure from this view is provided by Fisher, in [34], where epistasis is defined in a statistical sense. This definition proposes the search of statistically relevant interactions between alleles at different *loci* to explain a certain phenotype. This is the relevant definition to epistasis detection, which aims to find all interacting combinations of SNPs that may explain an observed phenotype.

To develop algorithms to solve the problem of epistasis detection, it is necessary to understand how epistasis is represented in computers. Table 2.2 depicts an example of a GWAS dataset, represented by two matrices. The first matrix represents the SNP data, where each line represents an array of SNPs from a patient (or sample) and each column represents a single SNP, encoded as previously explained in Table 2.1. The last column in the matrix represents the binary phenotype of each patient, with 0 representing the absence of a disease (control) and 1 representing the presence of a disease (case).

Table 2.2: GWAS dataset example

	SNP 1	SNP 2	SNP 3	SNP 4	...	SNP K	Label
Sample 0	1	0	1	1	...	2	0
Sample 1	0	1	2	1	...	0	1
Sample 2	2	0	0	2	...	1	1
...	...	...	...	...	...	...	...
Sample N-1	1	2	0	1	...	0	0

Interactions between SNPs are described as genotype combinations, where the number of SNPs in a combination represents the interaction order. In this context, interactions with order greater than three are considered to be high-order interactions. To simulate epistatic interactions, the most common strategy is to create penetrance tables. A penetrance table describes the probability of exhibiting a certain phenotype for each allele combination. To create epistasis datasets, generators like GAMETES [35], Toxo [36], and PyToxo [37] use these tables to express epistatic relationships.

To build penetrance tables and generate epistasis datasets using the aforementioned generators, it is necessary to specify Minor Allele Frequency (MAF) and heritability. The MAF of a SNP denotes the frequency in which the second most common allele occurs in a population. Epistasis datasets generators use the assumption of Hardy-Weinberg Equilibrium (HWE) [38] to calculate genotype frequencies, which considers that

$$\begin{cases} f(AA) = p^2 \\ f(Aa) = 2pq \\ f(aa) = q^2 \\ p + q = 1, \end{cases} \tag{2.1}$$

where $f()$ represents the frequency, p is the frequency of the dominant allele A and q is the frequency of the recessive allele a.

Heritability (h^2) describes the proportion of observable differences between individuals of a population that are explained by genetic factors. Heritability is described mathematically in [35] as

$$h^2 = \frac{\sum_i (P(D|g_i) - P(D))^2 P(g_i)}{P(D)(1 - P(D))} \tag{2.2}$$

$$P(D) = \sum_i P(D|g_i) P(g_i) \tag{2.3}$$

where $P(D|g_i)$ expresses the probability of exhibiting the phenotype having the genotype g_i, $P(g_i)$ is the probability of having genotype g_i, and $P(D)$ is the probability of expressing a phenotype in a population, also denoted as disease prevalence.

As penetrance tables describe probabilities, each penetrance value must be in the interval [0,1]. Therefore, not all combinations of heritability, prevalence, and minor allele frequency produce valid penetrance tables. Additionally, the generated tables may or may not exhibit marginal effects (i.e., the interacting SNPs may or may not influence the phenotype individually).

Table 2.3 provides an example of a valid penetrance table that exhibits no marginal effects. The marginal penetrance of each genotype is calculated as a dot product between the frequency vector and the corresponding penetrance vector.

As an example, the marginal penetrance of AA is given by

$$(0.5625, 0.375, 0.0625)\dot{(}0.729, 0.998, 0.760) = 0.5625 * 0.729 + 0.375 * 0.998 + 0.0625 * 0.760 = 0.778.$$

Calculating marginal penetrances for other genotypes would give the same result, not only for SNP A, but also for SNP B. Therefore, this table has no marginal effects.

Table 2.3: Example of a penetrance table without marginal effects

		SNP B			Marginal
	Genotypes	BB (0.5625)	Bb (0.375)	bb (0.0625)	Penetrance
SNP A	AA (0.81)	0.729	0.998	0.760	0.778
	Aa (0.18)	0.855	0.420	0.933	0.778
	aa (0.01)	0.751	0.942	0.006	0.778
Marginal Penetrance		0.778	0.778	0.778	

2.3 Epistasis Detection: Software Methods

To tackle epistasis detection, different software methods have been developed and can be broken down in several categories [5, 39], such as exhaustive methods, filtering methods, nature-inspired algorithms, Bayesian networks, and machine learning methods.

2.3.1 Exhaustive Methods

Exhaustive search methods analyze all possible combinations of SNPs to find the most accurate combination that explains the observed phenotype. The analysis of all combinations, while computationally complex, avoids finding possible sub-optimal solutions.

Exhaustive methods are based on the concept of contingency tables. A contingency table describes the frequency distribution of all possible genotypes. For example, for two SNPs, there are nine possible genotypes and the contingency table is a 3 x 3 matrix, as is exemplified in Table 2.4. The values presented on the table represent the frequency counts of cases or controls associated with a genotype. For example, n_{000} and n_{001} represent the number of controls and cases associated with the genotype 00, respectively. After building one contingency table, the frequency counts are evaluated with an objective function to measure the quality of the solution. This process is repeated for each possible pairwise interaction, which results in a computational demanding process for datasets with a high number of SNPs, specially when high order epistasis is considered.

Table 2.4: Contingency Table for Two SNPs

Controls	SNP2 = 0	SNP2 = 1	SNP2 = 2
SNP1 = 0	n_{000}	n_{010}	n_{020}
SNP1 = 1	n_{100}	n_{110}	n_{120}
SNP1 = 2	n_{200}	n_{210}	n_{220}
Cases	SNP2 = 0	SNP2 = 1	SNP2 = 2
SNP1 = 0	n_{001}	n_{011}	n_{021}
SNP1 = 1	n_{101}	n_{111}	n_{121}
SNP1 = 2	n_{201}	n_{211}	n_{221}

In [40], BOOST is proposed for exhaustive search of second order epistasis. By using a bit representation for SNPs (to reduce memory requirements) and a CPU @ 3.0 GHz, BOOST could analyze all possible SNP pairs of a dataset with 360 000 SNPs in 2.5 days. Another implementation, known as GWISFI, is proposed in [41] for exhaustive search of second order epistasis in a single GPU. In GWISFI, the GPU generates contingency tables for all SNP pairs in parallel, using multiple threads, stores them in global memory, and scores each SNP pair according to an user-defined statistic (e.g., Pearson's χ-squared test). As an example, this implementation could analyze a dataset of 5000 samples and 450 000 SNPs for second order epistasis in 12 minutes.

Nevertheless, scaling exhaustive search to high order interactions results in an intractable problem, as the search space of all combinations is given by

$$\frac{N!}{K!(N-K)!}, \tag{2.4}$$

where N is the number of SNPs and K is the interaction order. As K increases, the number of combinations increases exponentially, resulting in a high computational cost. For this reason, most exhaustive search attempts at epistasis detection do not go beyond third order interactions, evidencing the need for other methods.

2.3.2 Filtering

A common approach to reduce the computational complexity of epistasis detection is to discard a large subset of SNPs or prioritize small subsets according to some metric. To this end, filtering methods have been suggested [42].

One possible strategy is to use statistical evidence of single-SNP effects to prioritize SNPs that are deemed statistically significant. This approach inserts a bias into the analysis by assuming that relevant interactions occur only between SNPs that independently have some effect on the disease state. Some strategies include the use of statistical tests, such as Pearson's χ-squared test [43, 44], and Principal

Components Analysis (PCA) [45].

One of the most known methods is Multifactor Dimensionality Reduction (MDR) [46], a machine learning method that aims to define new attributes as functions of two or more attributes. MDR selects subsets of N SNPs within the pool of all SNPs. Next, the chosen N SNPs and their possible multifactor classes are represented. As an example, a set of two SNPs, each with three possible values, results in nine possible multifactor classes. For each class, the case-control ratio is estimated. If the ratio exceeds some threshold, the class is labeled as "high-risk"; otherwise, it is labeled as "low-risk". This strategy reduces the initial N-dimensional model to a one-dimensional model (one variable with two classes). MDR has succeeded in finding epistatic interactions in complex diseases, such as breast cancer [46] and amyotrophic lateral sclerosis [47]. Furthermore, since its first introduction, numerous extensions to MDR have been proposed to deal with large GWAS datasets [48].

Another strategy to filter SNPs is to employ feature selection algorithms to estimate SNP relevance for an observed phenotype. A common algorithm in the literature is the Relief [49], a filter algorithm that can capture feature dependencies by evaluating subsets of SNPs. Since its introduction, several variants have been proposed [50], as can be seen in Figure 2.1, which depicts the original $Relief$ algorithm, as well as some of its derivatives. In its original form, the Relief algorithm considers a set of n samples with p features. The algorithm runs for m iterations and start with a weight array of zeros with size p. At each iteration, a random instance (X_i) and its closest neighbors (by Euclidean distance) are chosen. The closest same-class instance to X_i is called 'near-hit', and the closest different-class instance is called 'near-miss'. The weight array is updated

$$W_i = W_i - (X_i - nearHit_i)^2 + (x_i - nearMiss_i)^2, \tag{2.5}$$

such that the weight of any feature decreases if it differs from that feature in nearby instances of the same class more than nearby instances of the other class, and increases in the reverse case. After m iterations, each element of the weight array is divided by m. This becomes the relevance vector. Features are selected if their relevance is greater than a threshold.

$ReliefF$ [51] finds a customizable number of nearest neighbors, instead of finding just two. Next, $IterativeRelief$ [52] is the first to attribute weights to neighbors according to their distance (the furthest the individual, the lower its weight for scoring a SNP) and limits the considered neighbors within a threshold distance. $I-Relief$ [53] is similar to $IterativeRelief$, but all neighbors are weighted, according to their distance. $SURF$ maintains the threshold from $IterativeRelief$ while assigning equal weights for the individuals below this threshold and ignoring those above it. $SURF*$ [54] uniformly weighs them with two different customizable weights, for individuals below and above said threshold. Finally, $SWRF*$ [55], $MultiSURF*$ [56], and $MultiSURF$ [57] are hybrids of $SURF$ [58] and $SURF*$. $SWRF*$ uses a sigmoid function for neighbors outside the threshold, $MultiSURF$ removes the scores for neighbors

outside the threshold, and $MultiSURF*$ introduces a customizable dead zone where no neighbors are scored. This family of algorithms, known as Relief-based methods, have successfully found epistatic interactions in simulated datasets [57, 58] and in some complex diseases, such as sporadic breast cancer [54].

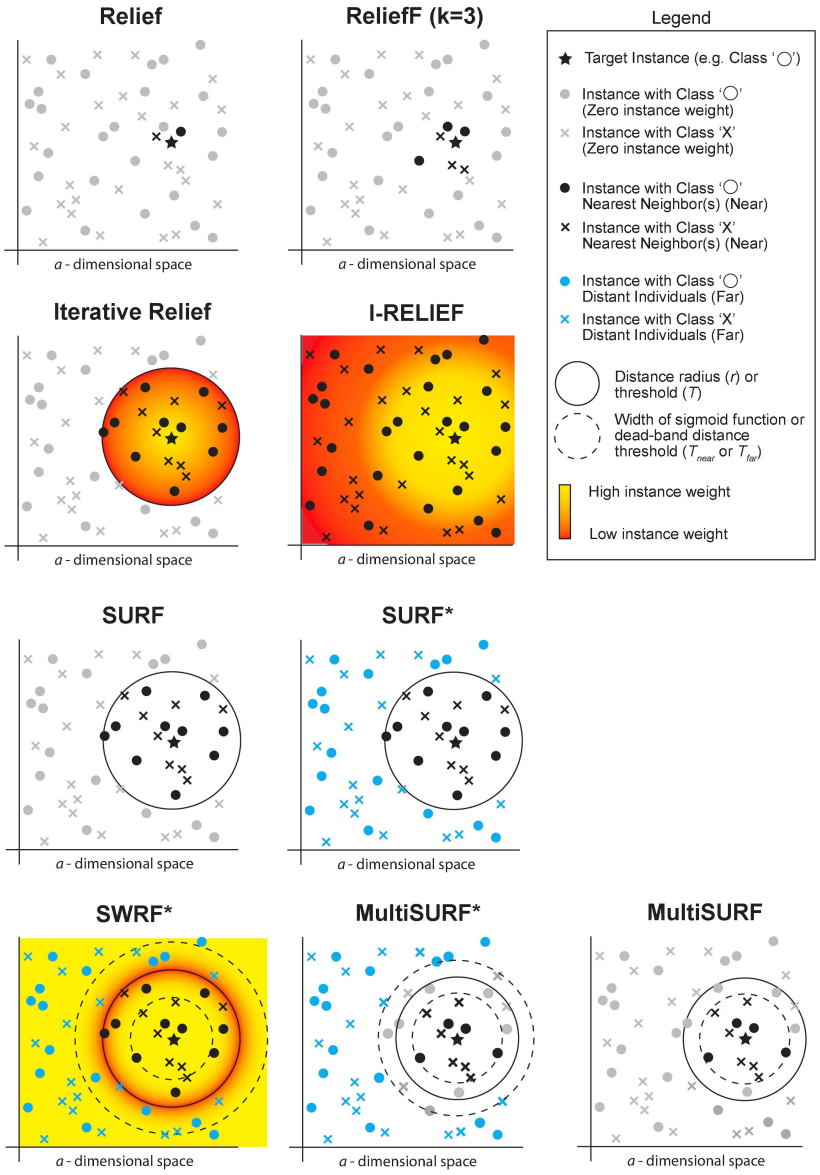

Figure 2.1: Summary of Relief-based Algorithms([50])

While filtering reduces the computational complexity, it introduces a bias into the detection process (for example, by testing only SNPs that have a high statistical correlation to the phenotype). Furthermore, it leads to a potential loss of accuracy, as not all combinations are evaluated for epistasis. One possible strategy to address these issues is by using random forests, a type of classifier that can find relevant SNPs without introducing a bias or doing exhaustive search.

2.3.3 Random Forests

Decision trees are a classifier model where each node represents a predictor variable and a path is a sequence of predictor variables from the root to the leaves. For epistasis detection, each node represents a SNP, as is represented in Figure 2.2. A tree-growing algorithm searches, in each step, for a SNP that optimally segregates the population. However, a shortcoming of this approach is the high dependence on the marginal effects of each SNP. To avoid the bias that comes from growing a single tree model, random forests were designed. Random forests create multiple decision trees to apply an ensemble procedure, which aggregates the predictions of all trees. The SNP set output is defined as the most important variable set of the random forest.

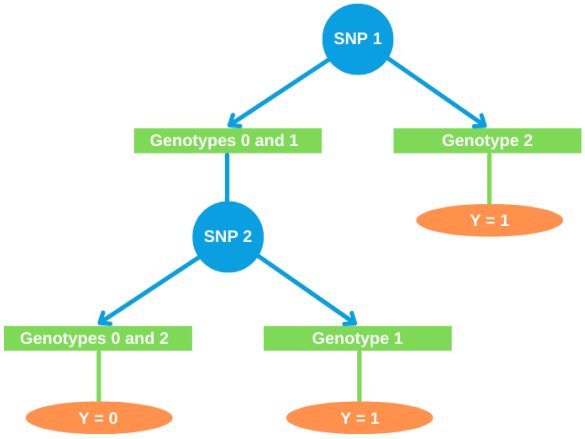

Figure 2.2: Decision Tree

In random forests, decision trees are built using a bootstrap strategy, i.e., N samples of K SNPs are selected randomly, with replacement, from a dataset with N samples. Individuals not drawn are classified as out-of-bag (OOB) samples. To build the decision tree, a random subset of SNPs is selected, ensuring low correlation between trees [59]. As an example, in [60], random forest is applied to select a subset of SNPs, based on Gini Index [61], that minimized classification error. The candidate SNPs were then statistically tested up to third order interactions and two interacting SNPS were successfully identified in an AMD dataset.

However, even if random forests find SNP associations that may be true interactions, there is no distinction between scenarios of interacting SNPs and the additive effect of several independent SNPs. Therefore, random forests lack a clear interpretation. A possible approach to tackle this issue is to employ Bayesian networks, as they offer an appealing and intuitive way to capture existing relationships between SNPs and observed phenotypes.

2.3.4 Bayesian Networks

Bayesian networks aim to find the causal relationships between sets of random variables and their conditional dependencies, while providing a compact representation of a joint probability distribution. The model is based on Bayes' theorem

$$P(N|D) = \frac{P(D|N)P(N)}{P(D)}, \tag{2.6}$$

where $P(N|D)$ represents the posterior probability distribution of the Bayesian Network N, given the data D, $P(D)$ is the probability of D, $P(N)$ is the prior probability of N (before observing the data), and $P(D|N)$ is the class-conditional probability.

The network is represented as a directly acyclic graph (DAG), where each random variable is a node on the graph and has a defined conditional probability distribution. If a causal probabilistic relationship exists between variables, a directed edge connects the nodes on the graph, as represented in Figure 2.3. In the case of epistasis detection, $X_1, ..., X_k$ represent SNPs and Y represents the observed phenotype.

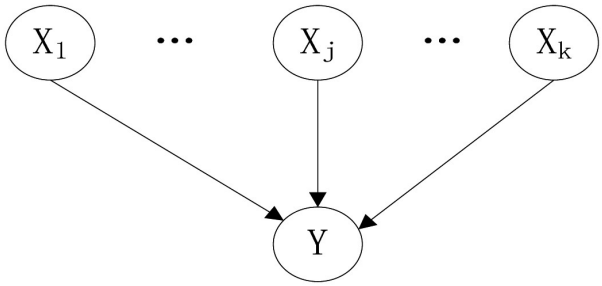

Figure 2.3: A Bayesian Network model between k-epistatic SNPs and disease state ([62])

In Bayesian networks, each variable is independent of its non-descendants, given its parents in the graph. Let X, Y, and Z be variables of the Bayesian network. If $P(X|Y,Z) = P(X|Y)$, then X is conditionally independent of Z, given Y (noted $X \perp Z|Y$). Therefore, the joint probability of *k* nodes is given by

$$p(X_1, X_2, ..., X_k) = \prod_{i=1}^{k} p(X_i|parent(X_i)), \tag{2.7}$$

where $parent(X_i)$ denotes the parent node of X_i. The probability of $P(D|N)$ can be computed, considering all variables in the DAG are discrete values, by

$$P(D|N) = \prod_{i=1}^{I} \left(\frac{\Gamma\left(\sum_{j=1}^{J} \alpha_{ij}\right)}{\Gamma\left(n_i + \sum_{j=1}^{J} \alpha_{ij}\right)} \prod_{j=1}^{J} \frac{\Gamma(n_{ij} + \alpha_{ij})}{\Gamma(\alpha_{ij})} \right), \tag{2.8}$$

where $\Gamma(\sum_{j=1}^{J} \alpha_{ij}) = (\sum_{j=1}^{J} \alpha_{ij} - 1)!$, α_{ij} is a parameter that refers to the knowledge about the number

of cases while the nodes take the corresponding i_{th} combination and j_{th} state, n_i is the number of samples associated with the i_{th} combination, and n_{ij} is the number of samples associated with the i_{th} combination and j_{th} state.

Taking an equal likelihood for all possible distributions in each Bayesian network model, $P(N)$ and $P(D)$ are set to constants, and $\alpha_{ij} = 1$, obtaining

$$P(N|D) \propto \prod_{i=1}^{I} \left(\frac{(J-1)!}{(n_i + J - 1)!} \prod_{j=1}^{J} n_{ij}! \right). \tag{2.9}$$

When the joint probability distribution in Equation 2.7 is assumed to be a Dirichlet distribution, the K2 score is defined as

$$K2 = \prod_{i=1}^{I} \left(\frac{(J-1)!}{(n_i + J - 1)!} \prod_{j=1}^{J} n_{ij}! \right). \tag{2.10}$$

Taking the logarithm of the previous expression, the K2 score can also be defined as

$$K2 = \sum_{i=1}^{I} \left(\sum_{b=1}^{n_i+1} log(b) - \sum_{j=1}^{J} \sum_{d=1}^{n_{ij}} log(d) \right). \tag{2.11}$$

Several studies for epistasis detection [13, 14, 16, 17, 62–65] refer to the K2 score in its logarithmic form. While Bayesian networks provide an intuitive representation of dependencies between variables, learning the network's structure amounts to a model selection problem, which is computationally intensive. Specific techniques have to be used to reduce the computational cost, such as nature-inspired algorithms.

2.3.5 Nature-Inspired Algorithms

Nature-inspired algorithms are a class of effective tools for solving optimization problems that are flexible and efficient and have become widely used for dealing with highly nonlinear problems and tough optimization problems [66]. While many nature-inspired methods exist [67], two classes are emphasized: Genetic Algorithms (GAs) and Swarm algorithms.

GAs are a class of methods that mimic the biological evolution process and have been employed to find epistatic interactions in complex diseases, such as AMD [68]. GAs begin with a random population of solutions to a problem that, over iterations, evolves via evaluation, selection, and mutation processes. GAs follow a common behavior to find solutions to a given problem:

1. Create a population of random solutions;

2. Test each solution using a fitness function;

3. Keep the best solutions and use them to generate new possible solutions. This new generation can be created using two mechanisms:
    - Crossover: choose two solutions and mix them (e.g. weighted average) to create new individuals. Crossover happens with a high probability;
    - Mutation: choose a solution and apply a random change (e.g. flipping a bit). Mutation guarantees that the search algorithm is not trapped in a local minimum, but occurs with a low probability.
4. If the termination criterion is fulfilled (e.g., a certain number of fitness evaluations has been reached), output the final population. Otherwise, jump back to step 2.

The population in genetic algorithms is usually evaluated by relying simultaneously on multiple objective functions, such as K2 Score and mutual information [65]. However, when considering more than one function (i.e. multi-objective genetic algorithm), it is difficult to find a solution that is optimal for all objectives. In general, a solution has a better performance on one objective and performs worse than other solutions for other objectives. This issue is generally tackled by finding the Pareto Set.

For two objective functions $f_1(X)$ and $f_2(X)$ and two possible solutions as X_1 and X_2, X_1 is said to dominate X_2 if one of the following conditions is satisfied:

1. $f_1(X_1) < f_1(X_2)$ && $f_2(X_1) < f_2(X_2)$,
2. $f_1(X_1) = f_1(X_2)$ && $f_2(X_1) < f_2(X_2)$,
3. $f_1(X_1) < f_1(X_2)$ && $f_2(X_1) = f_2(X_2)$.

If X_1 is not dominated by other solutions, X_1 is called a non-dominant solution. The Pareto Set is the set of non-dominant solutions, that are candidates to solve the multi-objective problem.

As a practical example, EpiMOGA [64] is applied to the study of Late Onset Alzheimer's Disease. The proposed multi-objective genetic algorithm seeks to find SNP interactions that minimize two functions: the Bayesian K2 Score (as a measure of correlation) and Gini Index (as a measure of statistical dispersion).

Similar to genetic algorithms, swarm optimization algorithms are nature-inspired and belong to the class of metaheuristic methods. Swarm optimization relies on the problem-solving capabilities that emerge from interaction between agents.

Ant Colony Optimization (ACO) is a commonly used algorithm that has been applied to epistasis detection, regarding, for example, Late Onset Alzheimer's Disease [69] and AMD [65]. In ACO, optimization problems are solved by simulating how ants discover food sources and communicate discoveries with other ants. As ants move, they leave pheromone trails. More ants on a trail means more pheromones, which motivates other ants to follow that trail instead of searching somewhere else.

In the case of epistasis detection, the search space comprises all SNPs in a dataset. The ants traverse the search space and build solutions consisting of SNP combinations, based on pheromone values and transfer rules. The probability of an ant k traversing from SNP i to SNP j is given by

$$P_k(i,j) = \begin{cases} R & if(q \leq T_0) \\ 1 & when j = rand(U_k(i)) if(q > T_0), \end{cases} \tag{2.12}$$

where $P_k(i,j)$ is the probability that an ant selects SNP j followed by SNP i, q is a number generated randomly from an uniform distribution in (0,1), and T_0 is a threshold to balance the convergence speed and avoid being trapped in local minimums. $U_k(i)$ represents the set of neighbors of SNP i that ant k has not yet visited and $rand(U_k(i))$ selects a random neighbor from this set.

R represents a selection strategy for the next SNP to add and is described by a probability distribution, depicted as follows

$$R = \begin{cases} \frac{\tau_{ij}^{\delta} \nu_j^{\beta}}{\sum_{u \in U_k(i)} \tau_{iu}^{\delta} \nu_u^{\beta}} & j \in U_k(i) \\ 0 & j \notin U_k(i) \end{cases}, \tag{2.13}$$

where τ_{ij} is the pheromone factor between SNPs i and j, ν_j represents prior information on SNP j, and δ, β are parameters. Pheromone factors are updated over iterations through positive feedback. The new factors are computed as

$$\tau_{ij_{NEW}} = \alpha \tau_{ij_{OLD}} + \Delta \tau_{ij_{OLD}}, \tag{2.14}$$

where α is the evaporation rate and $\Delta \tau_{ij_{OLD}}$ is an increment from ants that go from SNP i to SNP j.

MACOED [63] is an example of an epistasis detection method that relies on ACO to find promising combinations of SNPs for Late Onset Alzheimer's Disease. The ACO algorithm uses two metrics. The first objective function is the Akaike Information Criterion (AIC) score. This metric is derived from logistic regression, considering a model where the phenotypes are dependent variables and the genotypes are independent variables. The AIC score of the model is defined as

$$AIC = -2log(L) + 2d, \tag{2.15}$$

where $log(L)$ represents the maximized log-likelihood of the model, and d is the number of free parameters. The second objective function is the Bayesian K2 score. The algorithm aims to optimize these two metrics. As with genetic algorithms, finding solutions that optimize both metrics is difficult. To solve this issue, the Pareto Set is determined. Within the selected solutions, an exhaustive search with the statistical Pearson's χ-squared test is done. MACOED reports the solutions that have a p-value under the Bonferroni-corrected significance.

When nature-inspired algorithms are used, there is no guarantee that an optimal solution is found

on finite time and this solution may often depend on several parameters that require fine tuning. Furthermore, these algorithms find an optimal solution for a given data that may not generalize well in the presence of new data. A possible solution to these problems is to employ deep neural networks, given their flexibility in modeling complex relationships between variables that generalize well in the presence of new data.

2.3.6 Deep Learning

DNNs mimic the structure of the human brain and its ability to solve complex problems. Deep neural networks rely on training data to capture patterns and improve accuracy over time. These networks represent a directed graph in which neurons are connected by edges, whose direction represents the information flow within the network. The data to be processed enters the input layer and is processed through hidden layers until the final layer outputs the final result (classification). Each connection between neurons is attributed a weight to minimize prediction error.

Each individual node can be interpreted as its own linear regression model, composed of input data, weights, a bias, and an output. In the context of epistasis detection, the input layer takes a set of SNPs and the output is described as

$$y = \sum_{i=1}^{k} w_i f(x_i) + b = w_1 f(x_1) + ... + w_k f(x_k) + b, \tag{2.16}$$

where each x_i is a SNP, f is a non-linear activation function, and b and w_i are the "bias" and weights, respectively, to be estimated.

Training the network implies changing the weights in the network to optimize its performance for a given task. To train the network for data classification, the dataset to be used is split into three different subsets: training, test, and validation sets. The training set is used to train the model. The network's weights are randomly initialized and training is performed by observing the output and adjust the weights until classification is correct. After training, the validation set is used to evaluate the performance of the obtained model and the test set is used to evaluate the best model. When applied to epistasis detection, each input neuron represents a SNP. Each sample represents a set of genotype values and the correspondent disease state. Therefore, the model is trained to capture interactions between the input SNPs and the observed disease states to make accurate predictions on the phenotype.

The performance of a neural network is dependent on the data and the given task. Therefore, finding the most suitable network architecture is a key factor to maximize accuracy. Several architectures have been applied to epistasis detection [70–78], which will be discussed in this section.

Multilayer Perceptrons

Multilayer Perceptron (MLP), also known as fully connected feedforward networks, are artificial neural networks that consist of at least three layers: an input layer, a hidden layer, and an output layer. To the exception of the input nodes, every neuron in the network uses a non-linear activation function. Figure 2.4 presents the architecture of a MLP in the context of epistasis detection.

In [70, 71], MLPs are applied to epistasis detection in chronical dialysis and breast cancer datasets, respectively. In both cases, the network classified the datasets and determined the top 20 most significant SNPs, having identified true pairwise interactions. In [72], a MLP is applied to a subset of SNPs obtained by a clustering algorithm to identify epistatic interactions.

A recent implementation, described in [73], proposes the use of a MLP, with sparse regularization, to find epistatic interactions in protein datasets. Consider a protein that has N mutational sites (which can take binary values). It is possible to write a function, f, of these mutational sites as

$$f(X_1, X_2, ..., X_N) = \sum_{S \subseteq [N]} \alpha_S \prod_{i \in S} X_i, \tag{2.17}$$

where S is a subset of $1, 2, 3, \ldots, N = [N]$ and α_S is the Walsh-Hadamard transform coefficient associated with a combination in the set $X_1, X_2, ..., X_N$. This transform is a 2^N by 2^N matrix that is applied to 2^N numbers. In this case, the Walsh-Hadamard transform is applied to the 2^N combinations of a protein's mutational site, after being evaluated by a MLP, to obtain a function of the weights in the network. Finally, the L_1 norm is applied to promote sparsity within this function's coefficients. This approach works well for small values of N. For large values, a scalable alternative is proposed by sampling the sequences.

Despite the good results in protein datasets, the proposed implementation is unfeasible for SNP datasets. The Walsh-Hadamard transform operates on 2^N numbers and is therefore more suitable for binary variables than for SNPs. However, even if a binary representation exists for SNPs, using this method on modern datasets, given the sheer number of SNPs, is still infeasible, as simply sampling the sequences could not be enough to find promising SNP interactions.

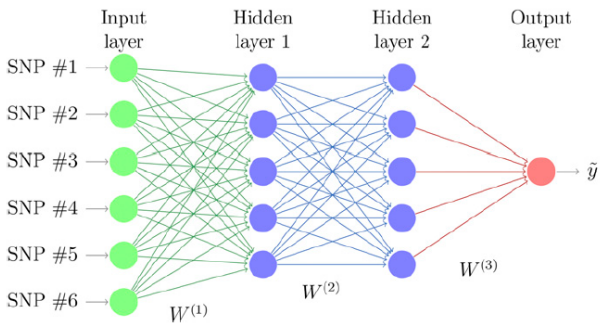

Figure 2.4: MLP Architecture Representation For Epistasis Detection ([21])

Another recent implementation, known as DeepCOMBI [79], attempts to use MLPs for epistasis detection, while providing explainability through Layerwise Relevance Propagation (LRP) [80]. LRP uses the network weights and activations generated by the forward pass to propagate the output back through the network up to the input layer and generate relevance scores for each input variable. With this strategy, the subset of the most relevant SNPs is identified and tested for possible epistatic interactions. This framework was tested in simulated and real datasets with reasonable success. However, because the proposed MLPs are dense networks, this framework scales poorly as the number of SNPs, and therefore it cannot be used for large epistasis datasets.

Stacked Autoencoders

Autoencoders are an unsupervised learning method that aim at feature extraction and dimensionality reduction. The autoencoder learns lower dimensional representations of the unlabeled input data. The architecture of an autoencoder consists of three layers, where the output, $\hat{x}$, should be as similar as possible to the input x. Given a dataset, the autoencoder learns a function such that

$$h_{W,b}(x) \approx x, \tag{2.18}$$

where W and b are the weights and bias in a given layer, respectively.

Training the network penalizes outputs different from the original input, thus learning a reduced representation for the data. However, simple autoencoders have a low representational power. To solve this limitation, Stacked Autoencoders (SAE) are proposed, as depicted in Figure 2.5. Autoencoders are stacked to enable greedy layer-wise learning where the l_{th} hidden layer is used as input to the $l+1$ hidden layer in the stack. The dimensionality of the data is decreased stack by stack, reducing noise and preserving the most important information.

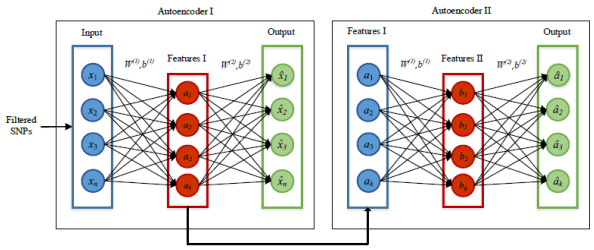

Figure 2.5: SAE Architecture Representation For Epistasis Detection ([74])

In [74], autoencoders were employed to detect epistatic interactions in extreme obesity. The autoencoder is applied as a dimensionality reduction strategy and the resultant low dimensional data is fed to a MLP to identify possible SNP interactions.

Convolutional Neural Networks

While MLPs are a powerful technique to model non-linear relationships for regression and classification problems, they are not the best option to manage spatial or temporal data. For situations where the input is distributed along a space pattern, such as in the context of image processing, Convolutional Neural Networks (CNNs) were proposed. Figure 2.6 depicts how convolutions work within the network.

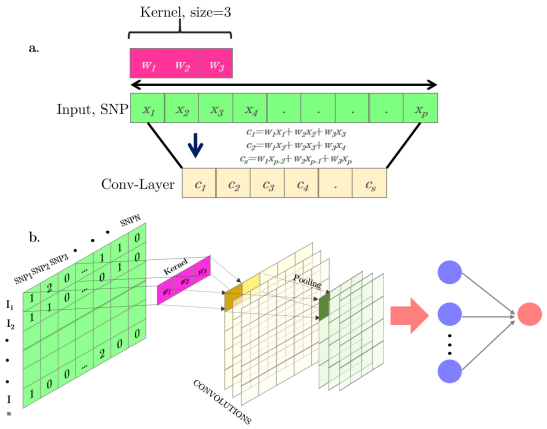

Figure 2.6: (**a**) One-dimensional Convolution Operation. (**b**) CNN Architecture Representation For A SNP Matrix ([21])

CNNs are composed by convolutional layers, pooling layers, fully connected layers, and normalization layers. In convolutional layers, a "kernel" (a multidimensional array of weights) is applied to the input data, followed by an activation function, to produce an output. Next, the pooling layer merges the "kernel" outputs, by taking the mean, maximum, or minimum of those values. Finally, the fully connected layer provides the final classification, based on the previously extracted features.

In [75], CNNs are applied to a PANCAN dataset, describing several types of cancer, to perform feature extraction and classification of the biomedical data. In [76], CNNs are applied to a dataset regarding Late Onset Alzheimer's Disease to extract population-level effects using large convolutional kernels.

Bayesian Neural Networks

In general, parameters on neural networks are assigned to single values or point estimates. Bayesian Neural Networks (BNNs) offer a different view on the weights by seeing them as probability distributions. Every parameter in the model is attributed a prior distribution, $P(W)$, which is usually a normal distribution. The main goal is to calculate the posterior distribution (after observing the data) for the weights, $P(W|X)$, where W is the network's weights and X is the data.

To find the posterior distribution, Bayes' law, as described in Equation 2.6, can be used. However, solving 2.6 requires a solution to an intractable integral that arises to find the probability of the data. To

overcome this issue, variational inference is employed. The posterior is approximated with a variational distribution, $Q(W)$, minimizing the Kullback-Leibler (KL) Divergence, defined as

$$D_{KL}(Q(W)||P(W|X)) = \int_{-\infty}^{+\infty} Q(W) log\left(\frac{Q(W)}{P(W|X)}\right). \tag{2.19}$$

From the KL Divergence between the distributions, it is possible to write

$$\underbrace{log(P(X))}_{\text{Evidence}} - \underbrace{D_{KL}(Q(W)||P(W|X))}_{\text{Objective}} = \underbrace{E[log(Y,X|W)]}_{\text{Classification Loss}} - \underbrace{D_{KL}(Q(W)||P(W))}_{\text{Regularization Loss}}, \tag{2.20}$$

where the right-hand side is called Evidence Lower Bound (ELBO). Minimizing ELBO maximizes the probability of the dataset being true, while minimizing the divergence between the variational distribution, $Q(W)$, and the posterior, $P(W|X)$, which is the objective.

It should be noted that BNNs and Bayesian networks are two different concepts. Bayesian networks are simple graphical models that describe probabilistic relationships between variables, while BNNs learn the probabilities and adjust the weights in the network to maximize the posterior probability distribution.

Bayesian neural networks have been applied to epistasis detection in simulated datasets [78] and in a tuberculosis dataset [77]. These neural networks, although promising, are, in practice, hard to implement and often need more training time to achieve acceptable precision, which may not be ideal from a time-to-solution perspective.

Transformers

Transformers are a novel architecture used in the scope of this master thesis that, to the best of our knowledge, is not yet present in the state of the art approaches for epistasis detection. The transformer was first proposed in [30] to handle long range dependencies in sequences and is the current state of the art technique for Natural Language Processing (NLP) [81]. Figure 2.7 depicts the general architecture of the transformer for sentence translation.

Consider an input sentence and its correspondent output in a different language. Each token (in the case of NLP, a token is a word) in the sentences is converted into a vector of size B (*Input Embedding* and *Output Embedding*). To account for the position of each word in its sentence, the positions are also converted into vectors of size B and added to the sentence embedding (*Positional Encoding*).

For the input sentence, these vectors are fed to the *Multi-Head Attention* layer, where attention is calculated. Attention is a mapping between a query (Q) and a set of key-value pairs (K,V). Q, K, and V are vectors obtained by linear transformations of the sentence embedding, which can be learned. A common attention mechanism is the scaled-dot attention, defined as

$$Attention(Q, K, V) = \underbrace{softmax(\frac{QK^T}{\sqrt{B}})}_{\text{Attention Score}} V, \tag{2.21}$$

where $\sqrt{B}$ is a normalization factor. The softmax function returns probabilities (attention scores) for each token. The higher the score for a word, the more important it is for the classification of the given data. Multiple heads are used to apply different linear transformations to the input embedding and learn different attention values for different word embeddings. The attention values are added to the sentence embedding and normalized (*Add & Norm*). These results are then fed to a fully connected layer (*Feed Forward*) and added to this layer's output before applying normalization again (*Add & Norm*).

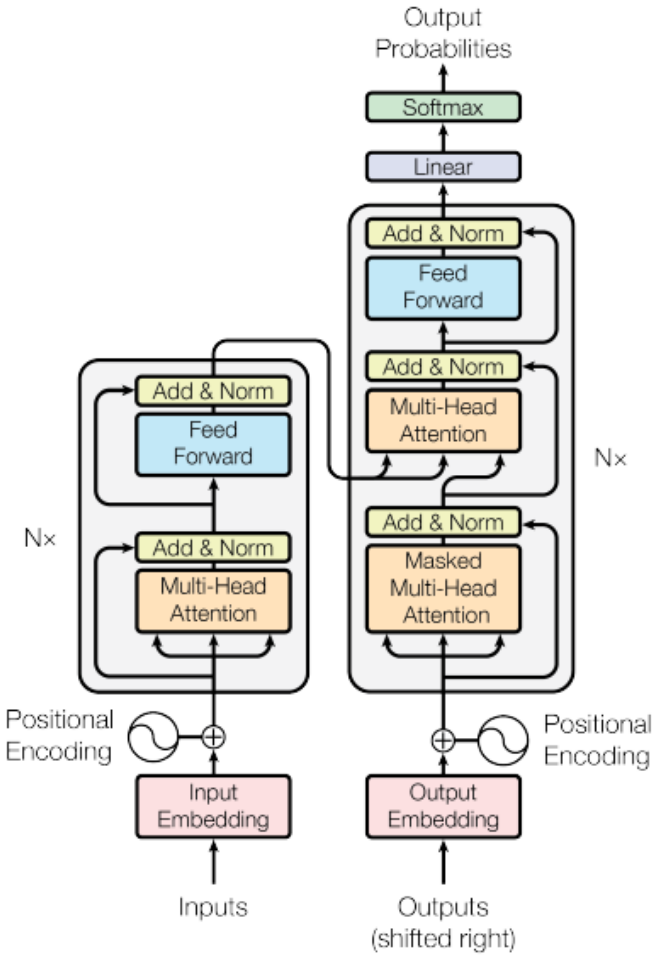

Figure 2.7: Transformer Architecture ([30])

For the output sentence, its embedding is fed to the *Masked Multi-Head Attention* layer, which masks some of the embedded inputs. The idea behind masking is to predict the next word using previous results without knowing the real word. The obtained attention scores are added to the sentence embedding and

normalized (*Add & Norm*). These results, along with the results obtained from the input sequence, are fed to another *Multi-Head Attention* layer, which maps words of the input and output sentences and finds relationships between them.

The final attention scores are added to the input of the *Multi-Head Attention* layer and normalized (*Add & Norm* layer). The result is fed into a fully connected layer (*Feed Forward*) whose output enters a linear layer (*Linear*). The final output of the transformer are probabilities for what should be the next word in the output sentence.

While the transformer provides good results in NLP problems, its performance is hindered by long sequences, as the necessary memory for the attention mechanism scales quadratically with the sequence length. Recent studies suggest that sparse transformers are enough to obtain the same accuracy as standard transformers [82] and are able to scale to long sequences [83,84]. Sparsity refers to a structure where most data is zero. In a neural network, sparsity can be classified as structured or unstructured. Structured sparsity constrains patterns in the weights to achieve memory representations with low overhead. On the contrary, unstructured sparsity allows for the pruning of arbitrary weights, albeit at the cost of increased overhead for index structures and loss of efficiency in execution of dense computation.

2.4 Epistasis Detection on Modern Computing Devices

While the approaches for epistasis detection are diverse, interactions beyond third order are not often explored due to the sheer computational size of the problem. A common approach to overcome this bottleneck is to employ specialized hardware to explore parallelism in epistasis detection. Table 2.5 depicts a series of works on epistasis detection using different devices. For each work, a brief description is provided, regarding device, year, biggest dataset, time to analyse all combinations, and the considered epistatic order. The presented works focus on two specific platforms: FPGAs and GPUs.

Table 2.5: Comparison Of Computing Devices For Epistasis Detection

Device	Year	Epistasis Order	#Samples	#SNPs	Time	Ref
GPU	2011	2	5003	351,542	1h20m	[11]
FPGA	2014	2	5000	500,000	4m	[18]
FPGA	2015	3	5000	10000	4h33m	[19]
GPU	2020	3	3200	6300	4m	[15]
GPU	2020	3	6400	40000	9h7m	[14]
GPU	2021	4	40000	400	20m	[17]
FPGA	2021	4	4000	2000	1h46m	[20]

FPGAs are programmable devices that offer a flexible platform for the implementation of custom hardware, tailored to a specific application, at a low development cost. Compared to GPUs, FPGAs

have relatively limited memory, bandwidth, and computing resources. However, they have the advantage of achieving similar performances with lower power consumption. Furthermore, FPGAs excel at applications where high levels of parallelism can be extracted, such as the creation of contingency tables in epistasis detection, as these devices enable models with highly flexible fine-grained parallelism. For these reasons, FPGAs are being successfully used to accelerate epistasis detection [18–20].

For second order interactions, [18] proposes an exhaustive search with the creation of contingency tables in parallel. This design is implemented on a RIVYERA S6-LX150 FPGA. This FPGA implementation analyzed a dataset with 500,000 SNPs and 5 000 samples in 4 minutes, achieving a speedup of 285x, in comparison to a CPU-only execution (2x Intel Xeon quad-core, 2.4 GHz). Furthermore, it achieves 98.8x better energy efficiency in comparison to the GPU.

For third order interactions, [19] proposes an exhaustive search with the creation of contingency tables and the use of mutual information as an objective function. This design is implemented on two FPGAs, a Virtex7-VX690T and a Kintex7-K325T. For a dataset with 10 000 SNPs and 5 000 samples, these FPGAs achieved a speedup of 363$\times$ and 181$\times$, respectively, compared to a CPU-only execution (Intel Core-i7 Sandy Bridge, 3.20 GHz with 6 cores).

A recent implementation, described in [20], is, to the best of our knowledge, the first attempt to tackle fourth order interactions using FPGAs. Using mutual information to analyze SNP combinations, this implementation processed a dataset with 2000 SNPs and 4000 samples in 1 hour and 46 minutes, using the Virtex-7 690T FPGA, 250 MHz. Furthermore, this architecture attained comparable performances to third-order GPU state-of-the-art approaches, with up to 8.2$\times$ better energy efficiency.

GPUs are designed for highly data-parallel applications with high arithmetic intensity. GPUs are organized as an array of Streaming Multiprocessors (SM). When a kernel is launched to the GPU for execution, thread blocks are scheduled onto an SM. The number of thread blocks that execute concurrently on an SM is referred to as the occupancy of the kernel. Higher occupancy is typically desirable, as thread-level parallelism can be exploited to hide the latency of memory and arithmetic operations. Due to their ability to explore parallelism, GPUs have proven to be successful in accelerating data processing in epistasis detection [12–17].

To perform second order epistasis detection, [11] implements BOOST [40], mentioned in Section 2.3.1, in a single GPU. Using a NVIDIA GTX 285 display card, GBOOST analyzed a dataset with 5003 samples and 351,542 SNPs in 1 hour and 20 minutes, resulting in a 44.8x speedup in comparison to BOOST.

Several GPU implementations exist to perform third order epistasis detection. As an example, in [15], MPI3SNP is presented. In the proposed implementation, SNPs are stored, using bitwise representations, as depicted in Table 2.6. SNPs are combined in groups of three to calculate genotypical frequencies and store them in contingency tables. An entropy-based algorithm is employed to rank the

interactions between SNPs. As an example, MPI3SNP analyzed an input of 6300 SNPs and 3200 samples in 4 minutes, using 8 NVIDIA K80 GPUs. By comparison, the same process would have taken over 3 days in a single CPU core.

Table 2.6: Binary Representation of one SNP

A	0	1	1	(...)	2	1
A0	1	0	0	(...)	0	0
A1	0	1	1	(...)	0	1
A2	0	0	0	(...)	1	0

In [14], the proposed implementation makes use of CPUs and GPUs. CPUs generate SNP combinations, which are sent to GPUs to build contingency tables and calculate the K2 Score for each combination. This implementation, on a system with an Intel Core i9-7900X and NVIDIA Titan V GPU, took 9 hours and 7 minutes to analyze all third order interactions in a dataset with 6000 samples and 40000 SNPs.

A more recent implementation, proposed in [17], is, to the best of our knowledge, the first attempt at fourth order epistasis using exhaustive search on GPUs. The generation of SNP combinations is done similarly to [14]. The contingency tables are partially constructed using third order interactions, as the calculation of frequency counts is less demanding than directly performing all calculations for a fourth order interaction. As an example, this implementation on a NVIDIA Titan RTX GPU @ 1.77 GHz with 4708 stream cores resulted in a 60x speedup with respect to CPU-only execution (2 Xeon Gold 6128 CPUs @ 3.4 GHz, each with 6 cores).

Some contemporary GPU architectures also have tensor cores, units specialized to matrix multiplication. In [13, 16], using the binarised representation of SNPs that was depicted in Figure 2.6, the contingency tables are built by using GPU's tensor cores to optimize the genotype counts. The emergence of tensor cores have led to the development of TPUs, which are designed as hardware accelerators for tensor operations in machine learning tasks, but have not yet been applied for epistasis detection.

To the best of our knowledge, IPUs have also not yet been applied to epistasis detection. The IPU is a recent massively parallel platform aimed at Machine Learning workloads. Its design premise is the efficient execution of fine-grained operations across a large number of parallel threads [29]. IPUs provide a large number of cores, each tightly coupled to a local memory to reduce latency. As each core only accesses its memory at a fixed cost and independently of access patterns, the IPU are efficient at executing applications with irregular computation and data accesses, as is the case of sparse data. Furthermore, for machine learning problems, such as natural language processing [26] and image recognition [28], IPUs obtained state of the art results. The IPU's good performance on machine learning problems, coupled with its efficiency on sparse data, makes it a promising platform to explore sparse

machine learning methods for high order epistasis detection that do not exist yet in the state of the art.

2.5 Challenges on Epistasis Detection

Table 2.7 depicts a comparison between several existing methods for epistasis detection. The landscape, while vast, tackles mostly second and third order interactions between SNPs. Improvements in hardware capabilities and algorithms have allowed for the search of high order interactions (such as fourth order, for example) using exhaustive methods, but only on datasets with hundreds to a few thousands of SNPs.

Table 2.7: Comparison Of Methods For Epistasis Detection

Method	Strategy	Year	Epistasis Order	#Samples	#SNPs	Ref
GWISFI	Exhaustive Search	2014	2	5000	450000	[41]
EpiMiner	Filtering	2014	3	2000	100	[85]
BNN	Deep Learning	2014	2	2000	1000	[77]
MACOED	Swarm Optimization	2015	2	1600	1000	[63]
PRF	Random Forest	2016	2	1282	39	[59]
FAACOSE	Swarm Optimization	2017	2	1368	309316	[69]
EpiACO	Swarm Optimization	2017	2	146	103611	[65]
HiSeeker	Filtering	2017	3	11950	423234	[43]
CNN	Deep Learning	2018	2	800	20531	[75]
Epi-GTBN	Genetic Algorithm	2019	2	146	1039	[68]
Deep Mixed Model	Deep Learning	2019	2	540	6970	[76]
MPI3SNP	Exhaustive Search	2020	3	3200	6300	[15]
SAERMA	Deep Learning	2020	2	1997	2465	[74]
EpiMOGA	Genetic Algorithm	2021	2	432	22164	[64]
HEDAcc	Exhaustive Search	2021	4	4000	2000	[20]
Fiuncho	Exhaustive Search	2022	6	2048	223	[86]

To find high order epistatic interactions in larger datasets, methods enhanced by machine learning are a reasonable approach. In particular, deep neural networks are a promising tool for epistasis detection, as they offer high flexibility in modeling complex relationships between variables. Furthermore, DNNs benefit from parallelism, as the existing data may be split in several batches and evaluated in parallel during inference. Therefore, the use of hardware that explore parallelism, such as GPUs, may accelerate training. However, two key issues arise in the use of neural networks.

First, the obtained model is difficult to interpret. Model interpretation refers to the extraction of feature importance from a dataset. To this regard, DNNs are black box systems, as there is no insight on the importance of features for the given task. In epistasis detection, while methods that rely on DNNs are often successful in classifying samples according to their phenotype, the biological interpretation is often

ignored. Therefore, the need arises for deep learning architectures that can easily identify interacting SNPs.

Second, as the scale of the given task increases, the networks also increase in depth and number of parameters to achieve equal or higher accuracy. A larger network leads to a high demand of memory and computation capacity. Recent studies suggest that a considerable proportion of the weights in DNNs are not necessary to achieve good performance [24, 25].

To compress neural networks and accelerate the training process, some approaches have focused on training sparse networks with less neurons and connections. Sparse models are desirable in deep learning to achieve higher accuracy and accelerate inference, as the necessary memory and operations are reduced [25]. However, sparsity poses a challenge for modern parallel architectures, as the performance of operations in sparse data varies drastically with the topology of nonzero values and level of sparsity. For traditional deep learning workloads, GPUs are the hardware of choice, and recent works have focused on developing efficient software to overcome the difficulties of sparse data processing [87, 88].

Although some works exist to address the issue of interpretation in DNNs and tackle epistasis detection [79], the use of transformers, as described in Section 2.3.6, can provide a more intuitive way of identifying relationships between SNPs, while also being scalable. Given the potential of sparse transformers for execution on large data [83, 84], choosing an adequate platform is relevant to achieve high performance. While GPUs are usually the hardware of choice, IPUs, as mentioned in Section 2.4, are tailored for machine learning workloads and are designed to be efficient on various kinds of memory access patterns. For these reasons, the application of sparse transformers in IPUs has great potential and are proposed in this thesis to target high order epistasis.

2.6 Summary

This Chapter started by providing the fundamentals on genetics to understand the importance of epistasis. Next, the state-of-the-art approaches for epistasis detection were examined, confirming the great variety of methods in the literature, while also depicting the existing bottlenecks. Modern computing devices were analyzed to understand how epistasis detection can be accelerated to reduce the computational time. Finally, the current challenges on epistasis detection were addressed.

Chapter 3

Methodology for High Order Epistasis Detection

The state of the art approaches for epistasis detection shows the potential of methods based on Deep Neural Networks to explore high order epistasis detection [21]. While this approach may be capable of predicting phenotypes based on SNPs, the main problem is the creation of black-box models, as it is difficult to understand how neural networks are predicting a given phenotype and how to find interacting SNPs [21]. Therefore, methods for network explainability are a necessary step to boost the potential of these methods for epistasis detection. Developing such methods is necessary to not only predict if a patient has a disease or not, but also to understand clearly which SNPs may be contributing for that classification.

This dissertation presents a novel methodology for interpreting deep learning methods and detecting interacting SNPs. Transformers are introduced as a promising deep learning technique that is yet to be applied for epistasis detection and that provides good interpretation measures for Natural Language Processing (NLP) tasks, as seen in the literature [89]. To understand how the transformer can be applied to tackle epistasis detection, its architecture is presented and modified, focusing on three parts. First, embeddings are studied to find meaningful SNP representations that the transformer can use for learning. Afterwards, it is necessary to understand how the transformer's attention mechanism works and how it can be used to assign scores for each SNP, providing network interpretation. Finally, because epistasis datasets can have thousands of SNPs and transformers do not scale well for large sequences, sparsity strategies are studied to mitigate the transformer's computational load.

To evaluate the performance of the transformer, simulated datasets are created to address several epistasis models under a wide range of values for heritability, minor allele frequencies, and interaction orders, emphasizing high order epistasis.

3.1 Framework for Epistasis Detection

Figure 3.1 depicts the complete framework for epistasis detection, based on transformers, which will be further elaborated in Section 3.2 to explain how to develop a transformer for epistasis detection. Given an input epistasis dataset, the positional embeddings are first calculated, using one of the three algorithms described in Section 3.2.1. Then, the dataset enters the transformer network, where each sample consists of an array of SNPs, plus the label, for which token embeddings will be generated and added to the positional embeddings. Attention is calculated using the SNP embeddings as a context and the label embedding as a token to predict. For each patient, the attention scores for each SNP are obtained and added together. After training, these scores are sorted by magnitude and the top 5%, 10%, and 25% are analyzed.

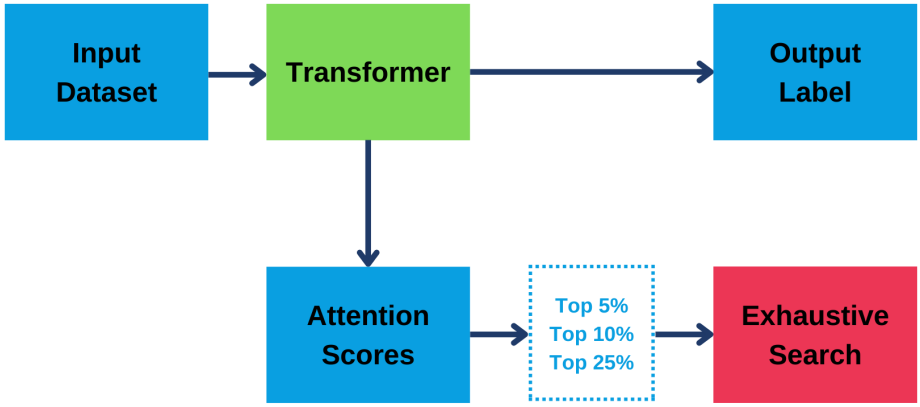

Figure 3.1: Framework For Epistasis Detection

Note that, for simulated datasets, because the interacting SNPs are known, evaluating the transformer's performance requires only checking if the interacting SNPs are in the top 5%, 10%, or 25% of attention scores. However, for real datasets, it is unknown whether interacting SNPs exist. Therefore, after identifying the SNPs that exhibit the top 5%, 10%, and 25% attention scores, an exhaustive search should be performed as an extra step to identify potential epistatic interactions within the SNPs filtered with the transformer.

This framework provides several advantages when compared to other deep learning techniques. First, it is a scalable framework. Unlike, for example, exhaustive methods, where the number of SNPs and patients is a bottleneck for evaluating epistasis, the proposed method is able to handle large

datasets. Second, it provides the necessary interpretation and explainability that was missing in other state-of-the-art approaches that apply machine learning methods to target epistasis detection. By means of attention scores, it is possible to quantify the relevance of each SNP in the prediction of a given phenotype and identify, similar to a filtering process. Third, this framework is configurable. For example, different embedding functions can be selected and the sparsity percentage on the attention module can be altered. Finally, note that, as Deep Learning models do not make assumptions on the datasets, the transformer should be able to identify interacting SNPs regardless of the epistatic order, meaning it is possible to explore high order epistasis.

3.2 Modifying the Transformer for Epistasis Detection

In Section 2.3.6, the transformer was presented as a state-of-the-art deep learning model that excels at NLP tasks. However, the original architecture, as depicted in [30], is proposed for sequence-to-sequence tasks. Given two sentences, the transformer learns relationships between words of both sentences. An example of a sequence-to-sequence task is sentence translation, for which the transformer was designed. For epistasis detection, however, the objective is to predict the phenotype of a patient (i.e., to predict whether it has the disease or not) and find which SNPs may be interacting to explain the observed phenotype. This task closely resembles another NLP problem for which the transformer can be applied, known as sequence modeling [90].

For sequence modeling, consider a sentence x with t words from a given vocabulary. The goal is to learn a probability distribution over a single word, $x[t]$, using the preceding words, $x[1 : t-1]$ as a context. This reasoning for NLP can be explored for epistasis detection, where the word to predict is a patient's phenotype, the patient's SNPs are the context, and the vocabulary is $\{0, 1, 2\}$. As will be explained in Section 3.2.2, each SNP will be assigned an attention score, providing a means to interpret what SNPs are considered relevant to predict a given patient's phenotype.

For sequence-to-sequence tasks, the transformer uses an encoder-decoder architecture, as is shown in Figure 3.2. Encoders aim to generate encodings that contain information about which inputs are relevant to each other. This is done using self-attention, which calculates the relevance of each token against all other tokens in a sentence. On the other hand, decoders receive the encoder's outputs, which incorporate contextual information, to generate an output sequence. Both layers have a feed-forward neural network for additional processing of the outputs and contain residual connections and layer normalization steps to keep gradients small during training.

For sequence modeling, it is shown in [91] that an encoder-only architecture is enough for this particular task, as the goal is to predict a token in a sentence using its context (and the encoder finds relationships between tokens) and not to generate sentences (which is the decoder's function). Therefore,

to apply transformers for epistasis detection, the decoder block is removed, while keeping the encoder without any structural changes. It will be necessary, however, to modify how the attention algorithm works, as the original algorithm is more focused on sequence-to-sequence tasks and not on sequence modeling. This will be further explained in Section 3.2.2.

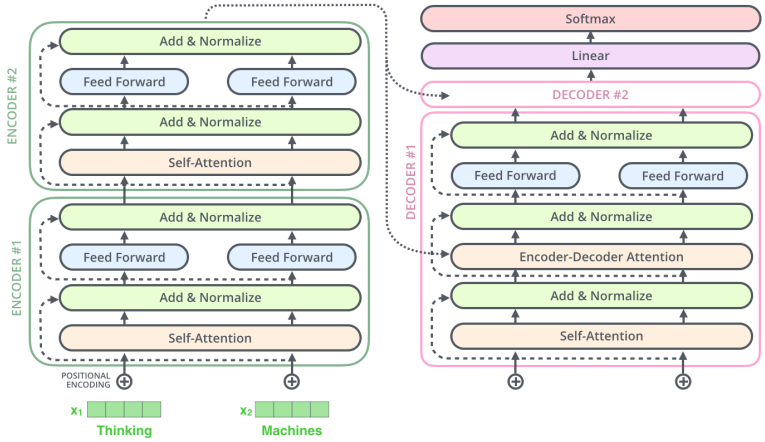

Figure 3.2: Encoder-Decoder Architecture

The overall structure of the remaining network is depicted in Figure 3.3. Given an input epistasis dataset, embeddings are calculated to represent the input SNPs of a sample (patient). These embeddings are fed to the encoder layer, which attempts to measure the relevance of each input SNP to predict the phenotype, using attention as the main algorithm. The encoder's outputs are then fed to dense layers before a sigmoid layer outputs the predicted label for the patient (0 or 1).

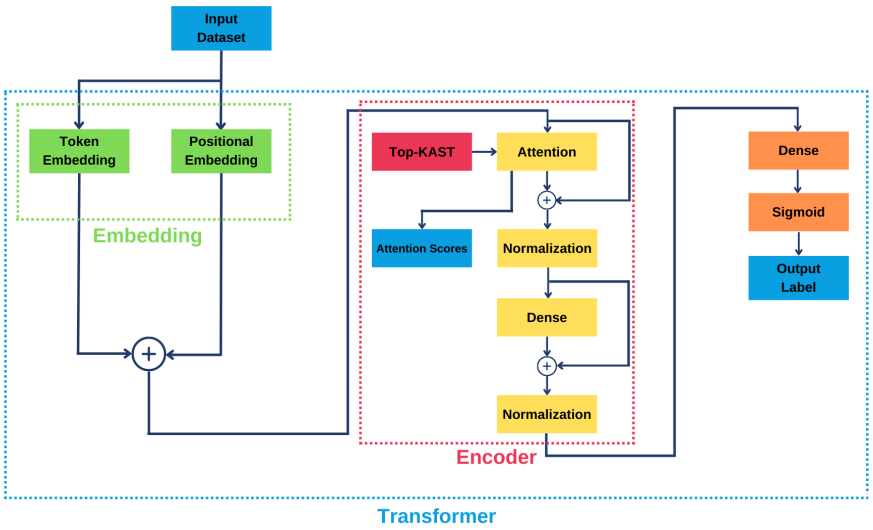

Figure 3.3: Transformer Architecture

A disadvantage of the proposed architecture lies in handling long sequences, due to scaling issues.

In particular, attention tends to scale quadratically with the sequence size, which poses a problem for epistasis detection, given that a dataset may have thousands of SNPs. A solution to this problem, as discussed in Section 2.3.6, is to employ sparse transformers, which are good for scaling to long sequences, while maintaining a reasonable performance. Consequently, to solve the scaling issues, a strategy for sparsity, Top-KAST [92], is employed to the attention module, as will be explained in Section 3.2.3, to enforce sparse patterns on the weight matrices that the attention algorithm requires.

Embeddings, as mentioned before, are the first module in the transformer. For epistasis detection, there is no categorical data, only numerical data, which could suggest that embeddings are not necessary. However, embeddings are kept in the network as a means to represent SNPs in a way that could potentially reveal SNP interactions. Therefore, to apply the transformer for epistasis detection, it is necessary to understand how to build meaningful SNP embeddings.

3.2.1 Embedding Representations

Because neural networks cannot interpret words, it is necessary to find a mapping between words and numbers in NLP tasks. While this is not a problem for GWAS datasets, which are already numerical, it may be advantageous to consider alternative SNP representations that make it easier to find SNP interactions during the transformer's training.

For epistasis detection, let us consider that each patient represents a sentence whose words are the SNP values (0, 1, or 2). A first approach to create SNP embeddings is to consider how many SNPs exist in a dataset. If N SNPs exist, then each SNP is associated to a vector of size N, where all entries are zero, except one. This process is called one-hot encoding. However, this approach has several disadvantages. First, it cannot represent possible similarities or relationships between SNPs. Second, the vector size increases linearly with the vocabulary size. Finally, as these vectors are highly sparse, neural networks struggle in finding meaningful information. A solution to these problems is to map each SNP to a dense vector, or embedding. A dense embedding allows for a richer SNP representation, as interacting SNPs may have similar embeddings, as opposed to the one-hot encoding.

The mapping between a SNP value and a vector is called the input or token embedding. However, the original transformer uses two types of embeddings. Because the same word in different positions in a sentence may have different meanings. it is also necessary to map the word's position to a vector. This is the positional embedding, which is added to the token embedding. For epistasis detection, as a patient is represented by an array with N SNPs, the SNP positions range from 0 to $N-1$. Therefore, each position will also be assigned a vector. These two embeddings (token and positional) are summed together to provide the complete embedding for a single word in a sentence.

Figure 3.4 depicts this process to obtain the complete embeddings for N SNPs, using embeddings of size D. Consider, as an example, the SNP in position 2, which, for a given patient, has a genotype

1, as highlighted in Figure 3.4. The positional embedding is a function that, maps the SNP position 2 to a vector $(0.3, -0.5, 0.03, 0.46)$, while the token embedding is another function that maps the genotype 1 to a vector $(0.1, 0.13, -0.2, 0.15)$. These two embeddings are summed together to obtain the complete embedding $(0.4, -0.37, -0.17, 0.61))$ for this SNP in the given patient. This process is performed until an embedding for all the SNPs and for the patient's label is obtained. The need for an embedding that represent the patient's label will be further explained in Section 3.2.2.

The embedding vectors are learned by the transformer and, therefore, change during training. For epistasis detection, as the vocabulary is small (3 genotypes) when compared to the number of positions in the sentence (N positions), the positional embeddings are more relevant than the token embeddings. Therefore, while token embeddings can be treated as parameters to be learned by the network, positional embeddings should be calculated *a priori* to ensure a meaningful SNP representation.

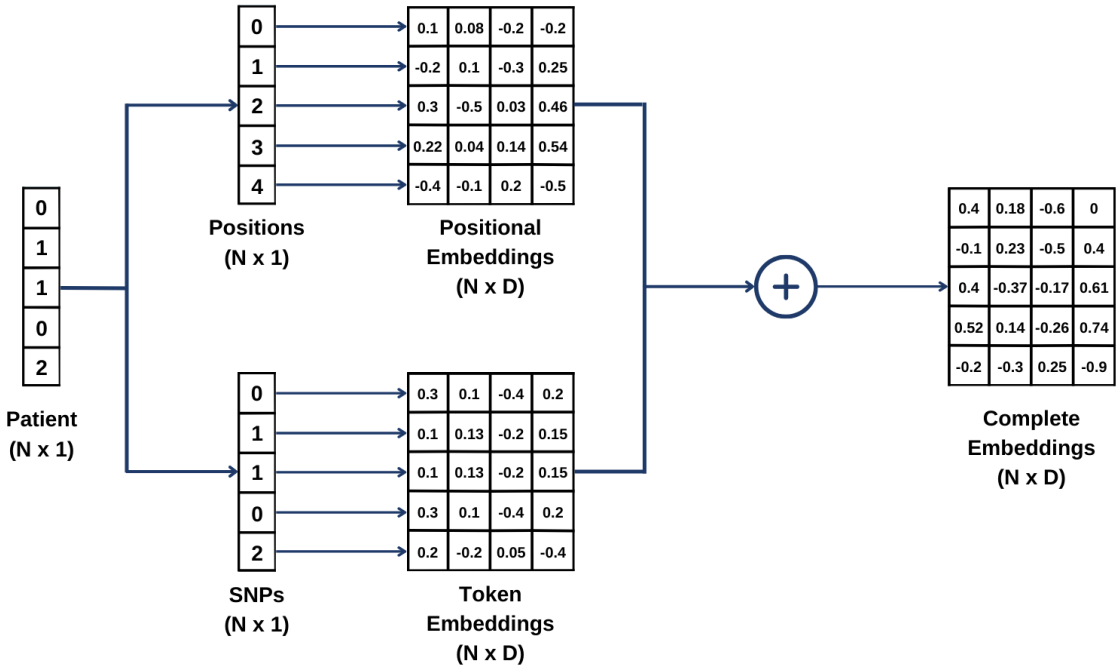

Figure 3.4: Embedding Workflow

In [30], positional embeddings based on sinusoidal functions are proposed to capture the fact that, for example, position 2 in an input sentence may be more related to position 3 than it is to position 10. In epistasis detection, interacting SNPs are not necessarily in adjacent positions for a given patient, so another strategy should be employed. Considering a dataset with P patients, each SNP, along with the label, can be thought as a point in a P-dimensional space, while an embedding can be considered a point in a D-dimensional space.

On epistasis datasets, P can potentially be a large number, but D should be small, given that the

transformer has layers with a quadratic complexity in D [30]. For this reason, positional embeddings should be obtained, for example, by dimensionality reduction techniques. Dimensionality reduction techniques have been used to visualize SNPs in lower dimensions [93] to find correlations between SNPs. Therefore, these techniques can be used to find meaningful SNP representations. Furthermore, the number of dimensions, D, can be selected freely. For these reasons, to build SNP embeddings, some of the most used dimensionality reduction techniques are explored.

Principal Components Analysis

Principal Components Analysis (PCA) [94] is a common dimensionality reduction technique that aims to find linear combinations of input variables, also known as principal components, by determining the maximum variance of the data. Assuming a dataset of N SNPs and P patients, the principal components can be written as

$$
\begin{aligned}
Y_1 &= a_{11}SNP_1 + a_{12}SNP_2 + ...a_{1N}SNP_N \\
Y_2 &= a_{21}SNP_1 + a_{22}SNP_2 + ...a_{2N}SNP_N \\
&... \\
Y_P &= a_{P1}SNP_1 + a_{P2}SNP_2 + ...a_{PN}SNP_N,
\end{aligned}
$$

where the first principal component, Y_1, points towards the direction of most variance in the data. If D principal components are calculated, then SNP_N is represented by $[a_{1N}, a_{2N}, ..., a_{DN}]$, which will be its positional embedding. For PCA, the choice of D is more intuitive. For example, D components can be calculated such that 90% of the data's variance is covered.

Principal components can be calculated from the eigendecomposition of the dataset's covariance matrix. Although PCA is a simple algorithm to build SNP embeddings, it is limited to linear projections, which means it cannot handle non-linear data well [95]. Therefore, non-linear dimensionality reduction techniques should also be considered as possible candidates to create SNP embeddings.

Locally Linear Embedding

Locally Linear Embedding [96] is a non-linear dimensionality reduction algorithm that aims at preserving the local geometry of the original data points (i.e., if the points were close in the high-dimensional space, the same points should be close in the low dimensional space).

To apply this algorithm, it is necessary to apply first K-Nearest Neighbors on the dataset, with a proper choice of K that accommodates the general structure of the dataset. Next, each point is reconstructed as a linear combination of its neighbors to minimize

The first step in Locally Linear Embedding is to consider, for a data point X_a, its K-Nearest Neighbors (based on Euclidean distance). After obtaining its neighbors (X_b), X_a is reconstructed as a linear combination of X_b to preserve the local properties of X_a. This is done by minimizing

$$E(W) = \sum_a (X_a - \sum_b W_{ab} X_b)^2,$$

where $E(W)$ is the reconstruction error and each weight, W_{ab}, represents the contribution of X_b to reconstruct X_a. The weights are selected such that $\sum_b W_{ab} = 1$ and that, if X_a and X_b are not neighbors, $W_{ab} = 0$. The points in low-dimensional space, Y_a, are chosen to also minimize the reconstruction error, with the weights W_{ab} previously calculated. The coordinates of these points can be calculated by eigendecomposition and represent the embeddings for each input variable. Note that, because K-Nearest Neighbors is used in this algorithm, the number of considered neighbors, K, can influence the quality of the embeddings.

Spectral Embedding

Spectral Embedding [97] is a nonlinear dimensionality reduction technique that, similarly to Locally Linear Embedding, aims to preserve the local geometry of the original data points. This technique is commonly used for graph structures and has already been applied in transformers for positional embeddings in [98].

Similar to Locally Linear Embedding, the first step is, for each SNP, to find its K-Nearest Neighbors and build an adjacency matrix, A, such that, if two SNPs are neighbors, they are connected by an edge with a positive value. This can be done using Gaussian weights or unitary weights. For epistasis detection, another possibility is explored, by setting the weight between two points as the inverse of the distance. The closer two points are, the higher the weights should be to ensure that low-dimensional representations of these points are also close [97].

After obtaining A, a diagonal matrix, Λ, is calculated, where the elements of the diagonal are the row sums of A. Finally, the Laplacian matrix, defined as $L = A - \Lambda$, is calculated. Performing an eigendecomposition of this matrix provides the low-dimensional representations of the input variables, which will be used as positional embeddings.

3.2.2 Attention Algorithm

After embeddings are calculated, the obtained SNP representations enter the attention module. Attention is a mechanism in the transformer architecture that uses contextual information (in the case of NLP, this information can be preceding text) for predicting the current token. The transformer, as is introduced in [30], is applied for sequence-to-sequence tasks, such as sentence translation, and attention

is calculated all-to-all tokens. This is known as self attention. However, the transformer may also be employed for sequence modeling, where the goal is to predict a token in a sentence, given the previous tokens (known as context tokens). For this task, a different strategy, known as single-query attention [99], is employed. This algorithm is tailored for sequence modeling, as it is employed to predict a word in a sentence, given all the previous words as context, and assigns scores to each context word. Similarly, in epistasis detection, the goal is to predict a phenotype, given all the SNPs in a patient and, using this algorithm, each SNP can be assigned a score for interpretation.

For single-query attention, the embedding of the current token to predict is mapped to a query vector and the embeddings of the context tokens are mapped to a key vector and a value vector through linear projections. Attention can then be calculated as

$$Attention(Q, K, V) = \underbrace{softmax(\frac{QK^T}{\sqrt{D}})}_{\text{Attention Score}} V. \tag{3.1}$$

where D is the embedding size, $Q \in \mathbb{R}^D$ is the query, $K \in \mathbb{R}^D$ is the key, and $V \in \mathbb{R}^D$ is the value.

The inner product QK^T can be interpreted as a measurement of how important is a context token to predict the current token. If the current token and a context token have similar embeddings, they are mapped to a similar query and key and, consequently, the inner product QK^T should be large, meaning that there could be a possible relationship between the two tokens. Conversely, if they have different embeddings, this product may be small, meaning that there is probably no existent relationship between the two tokens.

The next step is to apply the softmax equation, as follows:

$$Softmax(QK_t^T) = \frac{exp(QK_t^T/\sqrt{D})}{\sum_t exp(QK_t^T/\sqrt{D})}, \tag{3.2}$$

where $exp(.)$ denotes the exponential function. The output of a softmax function is a probability distribution over the context tokens. These probabilities, or attention scores, are then used to combine the value vectors. This calculation is summarized in Algorithm 3.1.

The algorithm's inputs are the current token embedding (e) and the context tokens embeddings (e_t) and the output is a vector representation of the token t and context projections combined ($\tilde{v}$). The parameters W_q, W_k, W_v represent the query, key, and value linear projections, respectively. The input e is mapped to a query, q, while the context, e_t, is mapped to a key, k_t, and a value, v_t, for every context token t. By applying the softmax function, an attention score, α_t, is obtained for every token t. Finally, these attention scores are linearly combined with v_t.

Algorithm 3.1: Basic Single-Query Attention

begin

Input: $e \in \mathbb{R}^{d_{in}}$, vector representation of the current token
Input: $e_t \in \mathbb{R}^{d_{in}}$, vector representations of context tokens $t \in [T]$
Output: $\tilde{v} \in \mathbb{R}^{d_{out}}$, vector representation of the token and context combined
Parameters: $W_q, W_k \in \mathbb{R}^{d_{attn} \times d_{in}}$, $b_q, b_k \in \mathbb{R}^{d_{attn}}$, the query and key linear projections
Parameters: $W_v \in \mathbb{R}^{d_{out} \times d_{in}}$, $b_v \in \mathbb{R}^{d_{out}}$, the value linear projection

$q \leftarrow W_q e + b_q$
$\forall t : k_t \leftarrow W_k e_t + b_k$
$\forall t : v_t \leftarrow W_v e_t + b_v$
$\forall t : \alpha_t = \frac{exp(q^T k_t / \sqrt{d_{attn}})}{\sum_u exp(q^T k_t / \sqrt{d_{attn}})}$
return $\tilde{v} = \sum_1^T \alpha_t v_t$

For epistasis detection, let us consider that the current token to predict is the patient's label. In that case, the label embedding is mapped to a query vector, Q, while the complete SNP embeddings are mapped to key and value vectors K and V, respectively. Equation 3.1 can then be applied to calculate attention. Figure 3.5 depicts the workflow for attention scores. During training, the attention scores for each sample (patient) are calculated and accumulated for analysis after training. As the attention scores represent a probability distribution, it is expected that, the higher the probability, the more important a SNP is to predict the phenotype of a given patient. For this reason, it is expected that interacting SNPs have high attention scores.

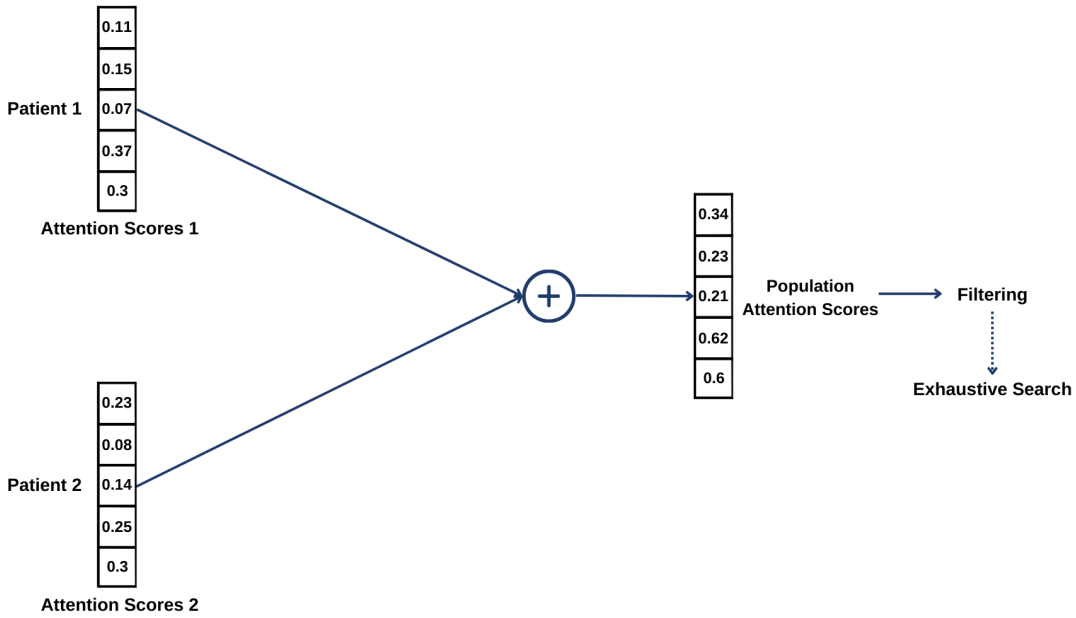

Figure 3.5: Workflow For Attention Score Calculation

After training the network, filtering the attention scores (e.g., by keeping the top 5% most relevant

SNPs) provide a means to find potential interacting SNPs. For simulated datasets, this filtering step is enough to find the interacting SNPs, given that they that are known. On real datasets, because the interacting SNPs are unknown, an exhaustive search should be performed as an extra step after the filtering to find the optimal SNP combination.

3.2.3 Sparsity in Attention

While the transformer has been shown to excel at NLP tasks, long sequences are a bottleneck of this network, as the necessary memory and computations scale quadratically with the sequence length [25]. This can be a problem when the transformer is applied for epistasis detection, as a single patient in a dataset may have hundreds of thousands of SNPs.

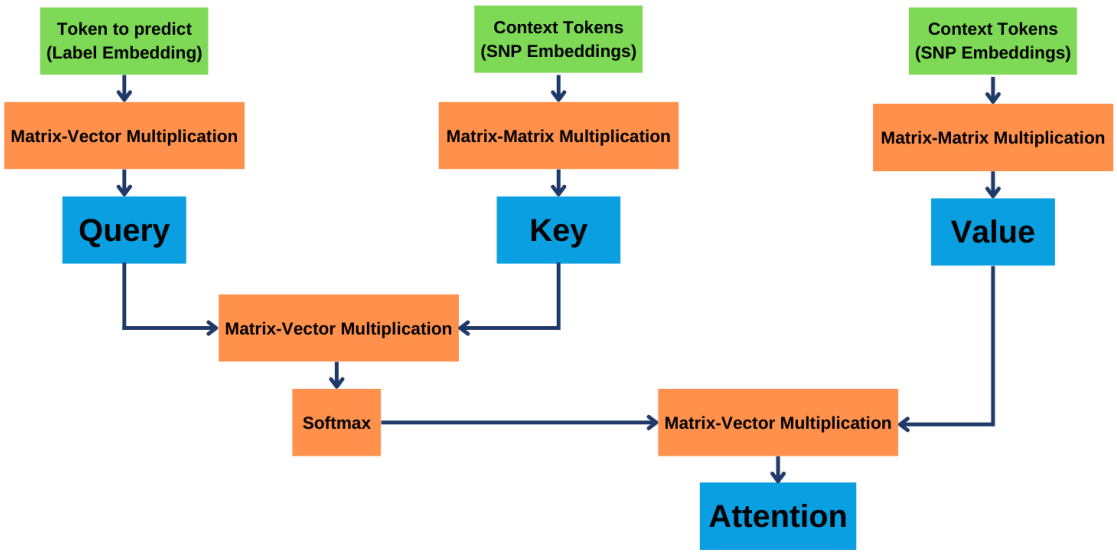

Figure 3.6: Attention Calculation

As mentioned in Section 2.3.6, recent research studies consider sparse transformers, which have been shown to obtain the same accuracy as standard transformers [82], while being able to scale to long sequences [83, 84]. The proposed strategies often try to sparsify attention, due to the complexity of the operations. Figure 3.6 depicts the necessary operations to calculate attention in the proposed methodology. The token to predict is represented by the label embedding and is multiplied by a weight matrix to obtain the query, which will also be a vector. On the other hand, the context tokens are a matrix of embeddings, with each row representing an embedding for one token. Creating the key and value requires multiplying the embedding matrix by a weight matrix, which also outputs a matrix. The inner product QK^T will then be a matrix-vector multiplication, which outputs a vector to which

softmax is applied. Finally, attention is a multiplication between the softmaxed vector and the value matrix. Therefore, in the proposed methodology, calculating attention would require three matrix-vector multiplications and two matrix-matrix multiplications.

In the state-of-the-art, the proposed strategies to sparsify the transformer are applied to architectures that employ self-attention mechanisms. Self-attention, as described in [99], applies attention to each token, considering that all other tokens represent the context. Self-attention is often employed in sequence-to-sequence tasks, but as this is not the case for the proposed epistasis detection methodology, other methods to sparsify the transformer should be considered.

As an example, the work proposed in [92] describes a strategy known as Top-KAST that has been tested for transformers. This method consists of selecting, for each training step, subsets of weights that correspond to the top-K weights by magnitude. Selecting weights by its magnitude provides an effective estimate of which parameters contribute the most to define the network's behavior, while also being inexpensive. To update the weights, a larger subset of weights is chosen to apply gradients. This keeps the gradients sparse, while allowing the weights with highest magnitude to change.

Figure 3.7 depicts the basic functional principle behind this. Consider a general weight matrix, C, with eight weights. Suppose that Top-KAST is used to select the top four weights. Then, in C, the four weights with highest magnitude are kept, while the others are set to zero. The resultant matrix, A, is now sparse. A is then used for operations in the neural network (e.g., multiplication with an input X) to generate an output Y. For backpropagation, in addition to updating the four weights of A, additional weights are updated to allow permutations between the weights with highest magnitude.

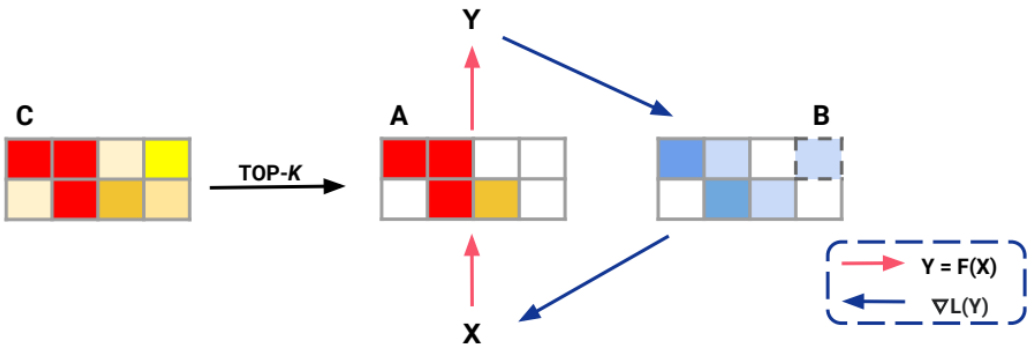

Figure 3.7: Top-KAST Strategy Illustration [92]

However, because it is unknown what weights could be more relevant for the network, in backpropagation, it is preferable that all the weights are updated, instead of just choosing a larger subset, allowing for permutations between the weights. Therefore, during the forward pass, the weight matrices are sparsified by selecting the weights with highest magnitude, but the gradients for backpropagation are dense, with every weight being updated. Another advantage of Top-KAST is that it is possible to choose how

sparse the weight matrices can be, and is therefore possible to explore the sparsity percentage as a hyperparameter for the neural network.

3.3 Hyperparameter Optimization

To implement deep learning models such as the transformer, hyperparameter optimization is a necessary step to maximize the model's performance. To evaluate transformer architectures, the first hyperparameters to consider are the number of inputs, the number of layers, activation functions, and learning rates. The number of inputs controls the amount of existent noise in a dataset and, therefore, influences the detection of interacting SNPs. For example, it is easier for a neural network to find two interacting SNPs in a set of 100 SNPs than in a set of 10000. The number of layers and activation functions influence the network's non-linearity. Finally, the learning rate influences the gradient updates during the backpropagation in the network. For low learning rates, the network converges slowly to a minimum loss, but high learning rates may lead to overshooting.

The aforementioned hyperparameters are general to any neural network model, but the proposed transformer also has hyperparameters that are unique. The first to consider is the embedding size, i.e., the size of the dense embedding which will be used to represent each SNP. As explained in Section 3.2.1, the transformer's complexity is quadratic with the embedding size, thus, this hyperparameter must be chosen carefully. Additionally, as it is unknown what would be the optimal algorithm to determine meaningful SNP embeddings, using different algorithms may yield interesting results and should also be considered a hyperparameter. To this end, the algorithms described in Section 3.2.1 are tested to verify which one provides the best embeddings.

Dropout percentage is also a hyperparameter to consider, as dropout layers are introduced in the network. High dropout percentages may hinder training by setting too many values to zero, but low dropout percentages may be used to avoid overfitting. Finally, as a consequence of the chosen strategy to sparsify the attention mechanism, it is possible to consider sparsity percentage as a hyperparameter to optimize. With low sparsity percentages, weight matrices may still be dense and the necessary computations are not reduced. On the other hand, high sparsity percentages could hinder the training process, as the weight matrices would have mostly zeros.

Table 4.2 provides possible ranges of values for some of these hyperparameters, considering state-of-the-art works [25, 30, 70, 71, 82, 91]. To perform hyperparameter optimization, a good strategy is to apply grid search to test all network architectures. However, the number of hyperparameters must be small. Applying grid search, considering all the hyperparameters of Table 4.2, would lead to a heavy computational load due to the sheer number of possible combinations and, for that reason, it is necessary to choose carefully which hyperparameters are more important to optimize. A more detailed

analysis of the chosen hyperparameters, as well as the range of values for each hyperparameter, is provided in Section 4.2.

Table 3.1: Hyperparameter Table

Hyperparameter	Range
Embedding Function	[PCA, Locally Linear Embedding, Spectral Embedding]
Embedding Size	$[32, 512]$
Activation Functions	[Relu, Tanh, Softsign]
Sparsity Percentage	$[0.1, 0.5, 0.7, 0.9, 0.99]$
Dropout	$[0.01, 0.05, 0.1, 0.3]$
Learning Rate	1×10^{-3}

3.4 Epistasis Modeling with Synthetic Datasets

To test novel methodologies for epistasis detection, a first step is to model epistasis with synthetic datasets, based on simulated penetrance tables. In this section, this topic is covered by studying the necessary parameters to generate penetrance tables and their potential impacts on the performance of deep neural networks, the mathematical models used to describe penetrance tables, and the interaction orders that can be achieved using state-of-the-art epistasis datasets generators.

3.4.1 Epistasis Parameters

To generate penetrance tables, it is necessary to specify two parameters. The first parameter is heritability, which, as explained in Section 2.2, is the percentage of phenotype differences that are explained by genetic variance. For this reason, different values of heritability have an impact on the performance of deep neural networks. For lower values of heritability, the chances of detecting interacting SNPs are lower, while for higher values, it is easier for a deep neural network to find interacting SNPs.

The second parameter is MAF, which is the frequency of the second most common allele in a population. This parameter also influences the performance of a deep neural network. For lower values of MAF, it is less likely for a SNP to take the value 2. In this case, it may be easier for a deep neural network to find interactions between SNPs and the observed phenotype. Likewise, for higher values of MAF, it may be harder to find those same interactions. Considering the influence of MAF and heritability, a range of values should be chosen for each parameter to cover as many epistasis scenarios as possible. The values adopted for each parameter are presented in Section 4.1.1.

3.4.2 Epistasis Models

In addition to MAF and h^2 values, a defined epistasis model is required to create penetrance tables. Models describe how epistatic interactions are mapped and determine the penetrance tables. Tables 3.2 to 3.5 describe the additive, multiplicative, threshold, and xor epistasis models, respectively, which are used to generate datasets. These models were first proposed in [100] and are commonly used to test the performance of epistasis detection algorithms.

In these tables, x represents the baseline probability of having a disease and y represents genotypic effects. The probability of having a disease increases by a factor of $(1+y)^z$, where z is a function of the genotype values. For example, in the additive model, z is the sum of genotypes. If a patient has a genotype AA (represented by 0) and a genotype bb (represented by 2), the odds of having the disease would be $x(1+y)^2$, as $z = 0+2$. Likewise, for the multiplicative model, z is the multiplication of genotype values; for the threshold model, z is 0, if at least one of the SNPs has genotype 0, and 1 otherwise; and for the xor model, z is the xor between the genotype values, which takes the values 0 or 1.

When using these models, not all combinations of MAF and heritability are possible. Because the equations used to find penetrance tables need these two parameters, depending on the model's complexity, it may lead to impossible solutions, meaning that MAF and heritability should be chosen carefully. Finally, note that penetrance tables defined according to the aforementioned models will exhibit marginal effects, meaning that SNPs interact individually with the phenotype.

Table 3.2: Additive Model Penetrance Table

Genotypes	AA	Aa	aa
BB	x	$x(1+y)$	$x(1+y)^2$
Bb	$x(1+y)$	$x(1+y)^2$	$x(1+y)^3$
bb	$x(1+y)^2$	$x(1+y)^3$	$x(1+y)^4$

Table 3.3: Multiplicative Model Penetrance Table

Genotypes	AA	Aa	aa
BB	x	x	x
Bb	x	$x(1+y)$	$x(1+y)^2$
bb	x	$x(1+y)^2$	$x(1+y)^4$

Table 3.4: Threshold Model Penetrance Table

Genotypes	AA	Aa	aa
BB	x	x	x
Bb	x	$x(1+y)$	$x(1+y)$
bb	x	$x(1+y)$	$x(1+y)$

Table 3.5: Xor Model Penetrance Table

Genotypes	AA	Aa	aa
BB	x	$x(1+y)$	x
Bb	$x(1+y)$	x	$x(1+y)$
bb	x	$x(1+y)$	x

3.4.3 Interaction Order

To tackle high order epistasis, it is desirable to create penetrance tables with various interaction orders. Using the PyToxo [37] library to generate penetrance tables, it is possible, for the models presented in Section 3.4.2, to generate penetrance tables for second to fifth order interactions.

Higher interaction orders are not explored, due to the complexity of the multiplicative model. As is explained in [37], PyToxo performs well for the multiplicative model up to fifth order, but cannot find higher order penetrance tables for most configurations of heritability and MAF. Therefore, to evaluate all models equally, only second to fifth order interactions are considered. After building the penetrance tables, the GAMETES [35] generator is employed to generate the simulated datasets that are used to train the transformer.

3.5 Summary

The main goal of this dissertation is to develop deep learning algorithms that are not yet present in the state of the art of epistasis detection and that can be easily interpreted, overcoming the usual black-box nature of these algorithms.

To this end, the architecture of the transformer is thoroughly analyzed, explaining its application for NLP tasks and how it can be applied to epistasis detection by devising SNP representations, based on embeddings, and interpretability measures, based on attention scores. Furthermore, methods for inducing sparsity on these networks are employed, as sparse transformers can achieve good performance. To test the proposed methodology under different epistasis scenarios, a wide variety of datasets is created with different parameters.

Chapter 4

Experimental Results

To validate the proposed methodology for network interpretation and SNP detection, a series of experimental tests are performed on the transformer to demonstrate its performance for epistasis detection. To this end, simulated datasets based on various parameters and models are created to evaluate the transformer's performance under multiple epistasis scenarios.

In this chapter, a hyperparameter optimization is performed to identify the optimal network to use for experiments. The experimental results are divided in three sections, each exercising different capabilities of the proposed framework. The first section focuses on evaluating different methods for SNP embeddings (using the algorithms presented in Section 3.2.1), while fixing the hardware, in order to analyze its impact on different epistasis models. The second section presents the transformer's results on the optimal network, which are then compared to state-of-the-art approaches for network explainability. Finally, to showcase the performance of the Graphcore IPU to train the transformer, this platform is evaluated on datasets with varying numbers of SNPs and patients and compared to NVIDIA GPUs and Google TPUs in terms of computational time.

To conclude the validation of the proposed methodology, the transformer is applied to a real breast cancer dataset. The interacting SNPs that are identified are compared to previous studies that applied exhaustive search algorithms to the same dataset.

4.1 Initial Configuration and Experimental Setup

In this section, the initial configurations used for the proposed experiments are presented. An overview of the created datasets is provided, as well as the chosen parameters and models. Next, the used metrics for the transformer's performance analysis are presented. Finally, the training parameters and the platforms that are employed to train the transformer are presented.

4.1.1 Datasets

As referred in Section 3.4, to create epistasis datasets, PyToxo is used as a tool to generate penetrance tables, while GAMETES uses these tables to generate simulated datasets.

The values used for MAF are [0.05, 0.1, 0.2, 0.5]. For each MAF value, the values used for heritability (h^2) are [0.01, 0.05, 0.2, 0.4]. These sets of values are similar to the ones used in MACOED [63]. To generate penetrance tables in PyToxo, the models described in Section 3.4.2 are used. For additive, multiplicative, threshold, and xor models, it is possible to generate penetrance tables for second to fifth order interactions, given the proposed values of MAF and h^2, although, for some combinations, it is impossible to generate penetrance tables for fifth order interactions using the multiplicative model.

For each combination of model, MAF, h^2, and interaction order, 100 datasets are created. For all datasets, the number of patients is set to 1600, with 800 controls and 800 cases, and the number of SNPs is set to 1000. Additionally, datasets with a range of patients from 400 to 6400 and a range of SNPs from 250 to 4000 are created for evaluation conducted in Section 4.5.

4.1.2 Performance Metrics

For classification tasks, it is typical to build a table known as a confusion matrix to analyze the performance of a supervised learning algorithm. This table is used to register the predictions made by the model and compare them with the ground truth. Ideally, the confusion matrix should only have elements in its diagonal, as these represent the number of correct predictions made by the model.

Table 4.1 is an example of a confusion matrix, where:

- True Positives (TP) is the number of positive samples (case) that are correctly predicted;
- True Negatives (TN) is the number of negative samples (control) that are correctly predicted;
- False Positives (FP) is the number of samples that are predicted positive (case), but the sample is negative (control);
- False Negatives (FN) is the number of samples that are predicted negative (control), but the sample is positive (case).

Table 4.1: Confusion Matrix

		Predicted Class	
		Negative (0)	Positive (1)
Real Class	Negative (0)	TN	FP
	Positive (1)	FN	TP

From the confusion matrix, it is possible to deduce performance metrics to evaluate the model, as described by Equations 4.1 to 4.4

$$Accuracy = \frac{TP + TN}{FP + FN + TP + TN}, \tag{4.1}$$

$$Precision = \frac{TP}{FP + TP}, \tag{4.2}$$

$$Recall = \frac{TP}{FN + TP}, \tag{4.3}$$

$$F1\ Score = \frac{2 * Recall * Precision}{Precision + Recall}. \tag{4.4}$$

Accuracy (Equation 4.1) is the proportion of correct predictions among the total number of examined samples. Precision (Equation 4.2) describes the ratio between correctly predicted cases among all the samples that were classified as cases, while recall 4.3) describes the ratio between correctly predicted cases among all the samples that are cases. Finally, the F1 score (Equation 4.4) is simply the harmonic mean between precision and recall.

These metrics allow to distinguish models that generalize well from overfitting models, specially for unbalanced data. For example, in a dataset with 90% controls and 10% cases, 90% accuracy may seem excellent, but if, for this case, precision is 0%, it would mean the model is only predicting the controls correctly, which makes it useless in practice.

4.1.3 Training Parameters

For all experiments, the number of epochs is set to 50 with a batch size of 32 samples. The network is trained on 90% of the total data, with the remaining 10% used as a test dataset to evaluate the performance metrics and the generalization capabilities of the transformer. To avoid overfitting, early stopping is employed, with a patience of 40 samples, meaning that if the validation accuracy does not improve in the following 40 samples, the training stops.

4.1.4 Setup

All the experiments are conducted on an IPU GC-200. All the networks are implemented in Python 3.7.3 and Tensorflow 2.4.0 and trained on the IPU. In Section 4.5, for hardware comparison, experiments are also conducted on an Nvidia A100 GPU and on a Google Cloud TPU-V3 Accelerator (using 8 cores).

4.2 Architecture and Hyperparameter Optimization

In Deep Learning, it is necessary to perform hyperparameter optimization to maximize a network's performance. Table 4.2 describes the used transformer architecture to optimize. The inputs are patients, characterized by SNPs and a label. First, an embedding layer maps the inputs to dense vectors, which enter the encoder. The encoder is characterized by an attention layer and feed-forward layer, interleaved with dropout layers (to avoid overfitting) and layer normalization (so that gradients do not explode during backpropagation). The encoder's outputs are averaged to obtain tensors with compatible size for the next layers. The final layers are dense, also interleaved with dropout layers. The last layer outputs the predicted label, according to a sigmoid function.

Table 4.2: Transformer Architecture

Layer	Type	Output	Activation	Dropout Ratio	Sparsity (%)
0	Input	(32, 1001)	-	-	-
1	Embedding	(32, 1001, 32)	-	-	-
2	Attention	(32, 1000, 32)	Relu	-	0.9
3	Dropout	(32, 1000, 32)	-	0.01	-
4	Layer Normalization	(32, 1000, 32)	-	-	-
5	Dense	(32, 1000, 32)	Relu	-	-
6	Dropout	(32, 1000, 32)	-	0.01	-
7	Layer Normalization	(32, 1000, 32)	-	-	-
8	Global Average Pooling (1D)	(32, 1000, 32)	-	-	-
9	Dropout	(32, 32)	-	0.01	-
10	Dense	(32, 32)	Relu	-	-
11	Dropout	(32, 32)	-	0.01	-
12	Dense	(32, 1)	Sigmoid	-	-

As explained in Section 3.3, the transformer has several hyperparameters to optimize. Grid search, while a good strategy to find the optimal network, works better on a small number of hyperparameters to avoid a combinatorial explosion. Consequently, it is necessary to prioritize the hyperparameters to evaluate. Three hyperparameters are considered for optimization: embedding functions (which influence how SNPs are represented), activation functions (which influence the network's training) and sparsity percentage (which influences the transformer's attention module and, therefore, the network's capabilities to find interacting SNPs).

The remaining hyperparameters are fixed. For the experiments conducted in this section, the number of inputs is kept fixed at 1000 SNPs, with 1600 samples. The dropout rate is fixed at 0.01, as is proposed in [21] that small values of dropout are preferable for training. Finally, the learning rate and embedding size are fixed at 0.001 and 32, respectively. Table 4.3 defines the grid search for the relevant hyperparameters of the proposed network architecture. The ranges for each hyperparameter were

chosen considering state-of-the-art works [25, 30, 70, 71, 82, 91].

Table 4.3: Hyperparameter Search Space

Architecture	Hyperparameter	Range
Transformer	Inputs	1000
	Embedding Function	[PCA, Locally Linear Embedding, Spectral Embedding]
	Embedding Size	32
	Activation Functions	[Relu, Tanh, Softsign]
	Sparsity Percentage	$[0.1, 0.5, 0.7, 0.9, 0.99]$
	Dropout	0.01
	Learning Rate	1×10^{-3}

For each combination of model (4 models), MAF (4 values), heritability (4 values), and interaction order (4 values), a dataset is sampled from the existent 100 datasets to test with the transformer. The number of datasets used to train each network would be 256 (4^4), but because no penetrance tables were found in some cases for the multiplicative model, the total number of datasets is 250. For each combination of activation function (3 functions), embedding function (3 functions), and sparsity percentage (5 values), a different network is obtained, which amounts to 45 networks. All possible networks are analyzed in terms of the number of datasets on which the interacting SNPs are found on the top 5%, 10%, and 25% of the attention scores, and in terms of accuracy. Figures 4.1, 4.2, and 4.3 show, for each evaluated hyperparameter, the dataset percentage on which the analyzed network found the interacting SNPs within the top 5%, 10%, and 25% of attention scores, as well as the average accuracy.

The first hyperparameter to analyze is the embedding function (PCA, Locally Linear Embedding, Spectral Embedding). The results depicted in Figure 4.1 show that the range of values of accuracy, as well as dataset percentage, is very similar for all the embeddings, without great improvements going from top 5% to top 25%. However, it is also possible to see that, for each embedding, there are networks for which the results clearly stand out, suggesting that the other hyperparameters could have more influence on the transformer's performance.

The next hyperparameter to analyze is the activation function (Relu, Tanh, Softsign). The results provided in Figure 4.2 show that different activation functions do influence the transformer's performance. While Relu and Softsign function seem to have similar results, some networks using the Tanh function achieve dataset percentages around 60% and above in the top 10% and top 25% charts, while also achieving good accuracy, suggesting that this activation function could be a good candidate for the proposed network.

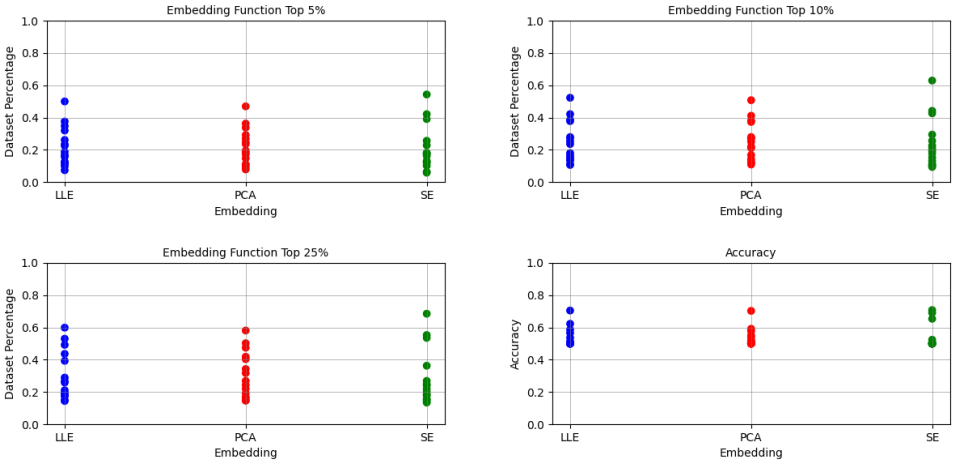

Figure 4.1: Transformer Embedding Analysis

The final hyperparameter to analyze is the sparsity percentage. The graphs presented in Figure 4.3 show the results for each considered percentage (10%, 50%, 70%, 90%, 99%). For 99%, the results show that the transformer finds interacting SNPs on a small percentage of the training datasets, with low accuracy. This is an expected result, as, for a given weight matrix in the attention layer, 99% of its values are zero, resulting, as a consequence, in a model with a poor performance. On the other hand, low sparsity (10%) exhibits a slight better performance than 99%, while still keeping a low accuracy, because, for this percentage, the weight matrices are still dense, which hinders the training due to noise. The intermediate values for sparsity (50% and 70%) provide better results in terms of accuracy and dataset percentage, while 90% as the sparsity percentage provides both better accuracy and better dataset percentage, when compared to network that employ other sparsity percentages.

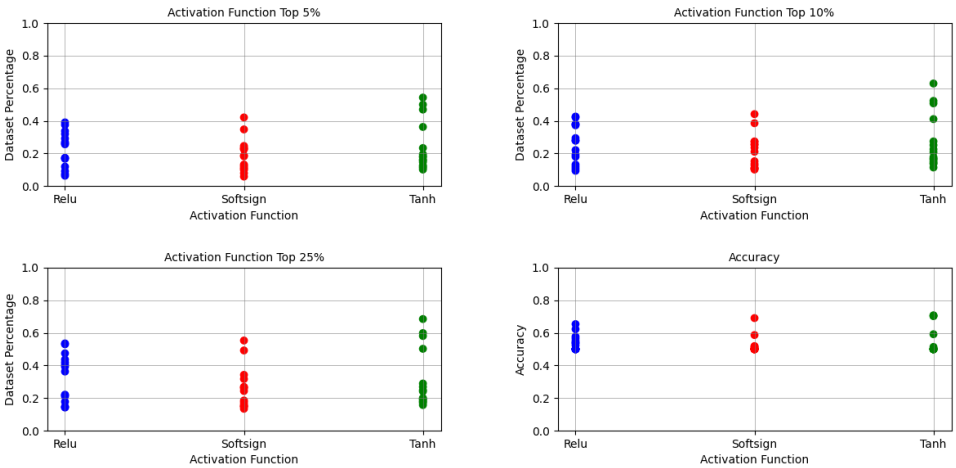

Figure 4.2: Transformer Activation Function Analysis

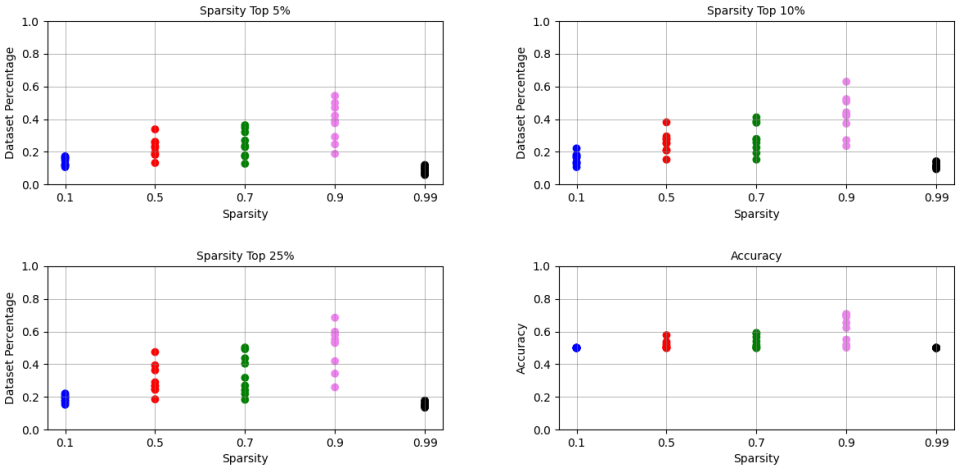

Figure 4.3: Transformer Sparsity Analysis

Table 4.4 presents the hyperparameters for the 5 networks that display the best results out of all analyzed networks (45). An interesting result to note is that the top 3 networks use the same activation function (hyperbolic tangent) and sparsity percentage (90%), while changing the used embedding function, which supports the previous analysis done for each hyperparameter. Furthermore, the optimal network, which will be used in Section 4.4 for comparison with state-of-the-art approaches, uses Spectral Embedding as an embedding function, hyperbolic tangent as an activation function, and applies a 90% sparsity pattern on the attention module.

Table 4.4: Top 5 Networks

Ranking	Embedding	Activation	Sparsity	Top 5%	Top 10%	Top 25%	Accuracy
1	SE	Tanh	0.9	0.543	0.63	0.685	0.708
2	LLE	Tanh	0.9	0.5	0.523	0.599	0.705
3	PCA	Tanh	0.9	0.469	0.508	0.581	0.703
4	SE	Softsign	0.9	0.421	0.442	0.553	0.691
5	SE	Relu	0.9	0.391	0.426	0.536	0.691

4.3 Embedding Comparison

Given that it is unknown which algorithm could be better to represent SNPs, exploring the performance of all algorithms may yield interesting results. To this end, the top 3 networks found in Section 4.2 are selected to run all datasets, given that all parameters are fixed for these networks, except the embedding algorithms. Figures 4.4 to 4.6 present the obtained results. In these images, each row represents a combination of heritability and MAF, each column represents model and interaction order, and each number represents the percentage of datasets on which the transformer found all interacting SNPs

within the top 25%, for a given combination of MAF, heritability, model and interaction order. For each algorithm, an analysis of the transformer's performance for each epistasis model (additive, multiplicative, threshold, and xor) is provided.

Starting with PCA, the results for the top 25%, as depicted in Figure 4.4, suggest that PCA is a good strategy to create SNP embeddings for additive models. In fact, considering all datasets for the additive model, using PCA, the transformer can find all interacting SNPs in 90.6% of the datasets. This behavior is not unexpected, as PCA works by finding linear combinations of the input variables (in this case, the SNPs). An additive model works in a similar fashion and, consequently, it is expected that building SNP embeddings with PCA provides good results for this particular model. However, for the multiplicative, threshold, and xor models, PCA is a poor choice, as the transformer finds the interacting SNPs in 27.6%, 35.9%, and 6.25% of the datasets, respectively.

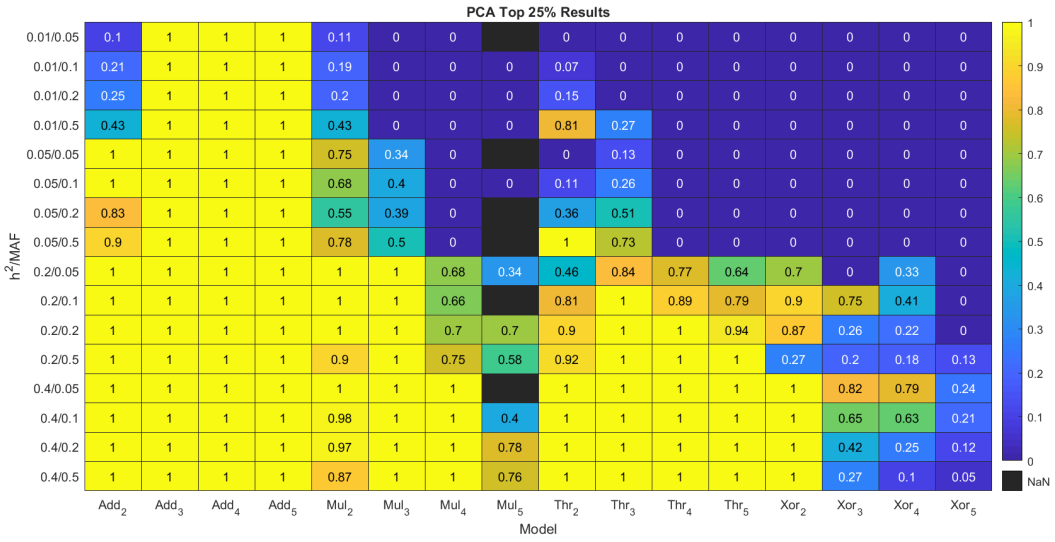

Figure 4.4: PCA Embedding Top 25%

Given that Locally Linear Embedding and Spectral Embedding are both non-linear dimensionality reduction techniques, an increase in the transformer's performance could be expected for the multiplicative, threshold, and xor models. The results for Locally Linear Embedding, provided in Figure 4.5, show that, for additive and threshold models, the transformer finds all interacting SNPs for 71.9% and 34.3% of all datasets, respectively, resulting in a worse performance, when compared to PCA. On the other hand, for the multiplicative and xor models, the transformer can find all interacting SNPs for 37.9% and 12.5% of all datasets, respectively. The biggest improvement is in the xor model, where using this embedding doubles the transformer's performance, in comparison to PCA.

On the other hand, Spectral Embedding seems to work with reasonable performance across all epistasis models. The results provided in Figure 4.6 show that, for the additive model, the transformer

finds all interacting SNPs in 81.8% of all datasets, which is better than Locally Linear Embedding, but worse than PCA. For multiplicative, threshold, and xor, the transformer finds all interacting SNPs in 60.2%, 58.8%, and 52.1% of all datasets, respectively, which outperforms both PCA and Locally Linear Embedding. Again, the biggest improvement is in the xor model, where using Spectral Embedding results in a 40% improvement, when compared to Locally Linear Embedding, and a 46% improvement, when compared to PCA.

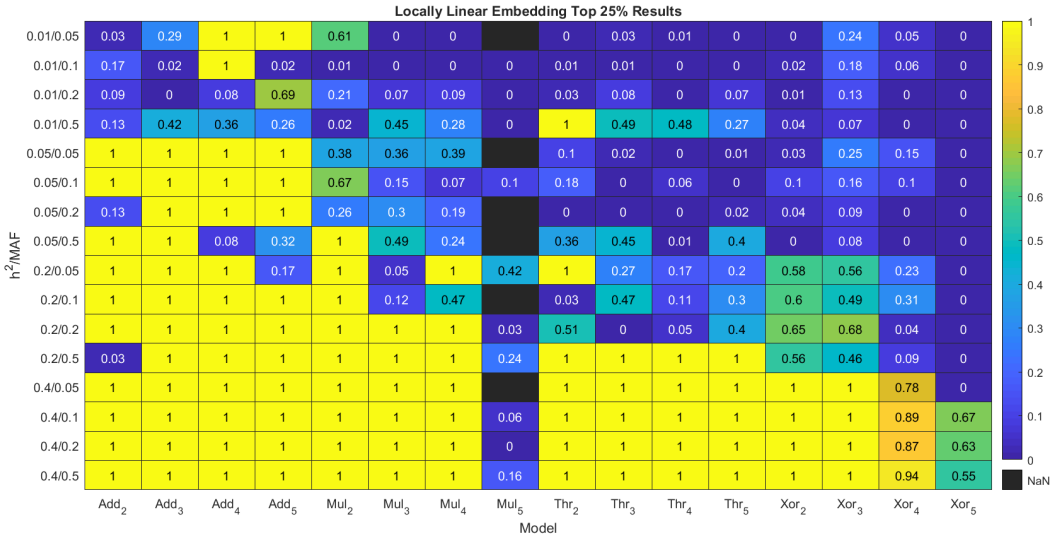

Figure 4.5: Locally Linear Embedding Top 25%

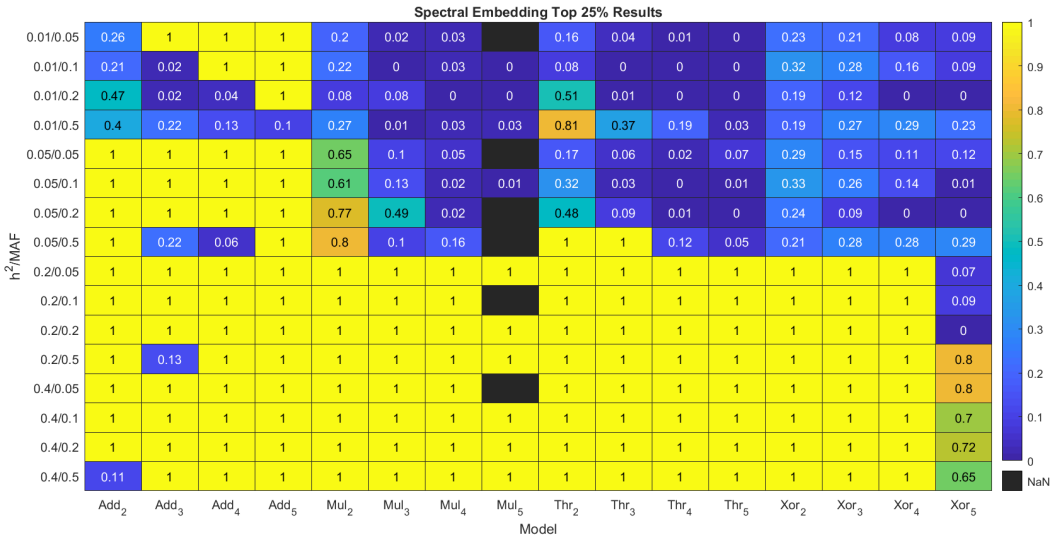

Figure 4.6: Spectral Embedding Top 25%

Table 4.5 summarizes the obtained results for each epistasis model, using the aforementioned em-

bedding functions.

Table 4.5: Embedding Results

	Dataset Percentage		
	PCA	LLE	SE
Additive (6400 datasets)	**90.6%**	71.9%	81.8%
Multiplicative (5800 datasets)	27.6%	37.9%	**60.2%**
Threshold (6400 datasets)	35.9%	34.3%	**58.8%**
Xor (6400 datasets)	6.25%	12.5%	**52.1%**

4.4 Comparison with State of the Art Approaches

To compare the performance of the proposed transformer framework in terms of interpretation, the work described in [79] is used, referred herein as DeepCOMBI. In this work, a MLP is used to predict phenotypes in GWAS datasets. For network interpretation and SNP detection, LRP [80] is employed to find the most relevant SNPs. LRP uses the network weights and activations generated by the forward pass to propagate the output back through the network up to the input layer and generate relevance scores for each input variable. Table 4.6 describes the architecture of the employed MLP, characterized by two dense layers of 64 neurons, regularized with L1 and L2 norms to avoid overfitting, interleaved with two dropout layers with a dropout percentage of 0.3. The final layer, using a softmax activation function, outputs the probabilities for each possible label.

Table 4.6: MLP Architecture

Layer	Type	Input	Output	Activation	Dropout Ratio	L1 Regularizer	L2 Regularizer
0	Input	(40, 1001)	(40, 1001)	-	-	-	-
1	Dense	(40, 1001)	(40, 64)	Relu	-	0.0001	0.000001
2	Dropout	(40, 64)	(40, 64)	-	0.3	-	-
3	Dense	(40, 64)	(40, 64)	Relu	-	0.0001	0.000001
4	Dropout	(40, 64)	(40, 64)	-	0.3	-	-
5	Dense	(40, 64)	(40, 2)	Softmax	-	0.0001	0.000001

As mentioned in Section 4.1.1, for each combination of model, MAF, h^2, and interaction order, 100 datasets are created. To compare the transformer and DeepCOMBI in terms of execution time, the necessary time to train each network using 100 datasets is measured. Table 4.7 shows the obtained results for each network. For 100 datasets, the transformer needs 14 minutes and 23 seconds for training, while DeepCOMBI needs 54 minutes and 32 seconds, meaning that a 3.79x speedup is obtained when the transformer is employed for training epistasis datasets.

Table 4.7: Transformer and DeepCOMBI Execution Times for 100 Datasets

	Execution Times
Transformer	14m23s
DeepCOMBI	54m32s
	Speedup = **3.79x**

Next, to understand the limitations of the transformer in terms of interpretability and compare it with DeepCOMBI, for each epistasis model (additive, multiplicative, threshold, and xor), results regarding the top 5%, 10%, and 25% of attention scores are obtained, as well as the results regarding the performance metrics (accuracy, precision, recall, and F1 score). For each model, the most relevant results for the analysis are presented, while the remaining results for relevant figures of merit can be found in Appendix A.

4.4.1 Additive Models

For additive models, Figures 4.7 to 4.9 show the percentage of datasets where interacting SNPs had attention scores that were on the top 5%, 10%, and 25% of all scores. These figures show side by side the results for the transformer (left) and for DeepCOMBI (right). Each number represents the percentage of success (ranging from 0 to 1) in finding interacting SNPs for a given combination of MAF, heritability, and interaction order. The total number of training datasets is 6400, as there are four interaction orders, sixteen combinations of MAF and heritability, and 100 datasets for each combination.

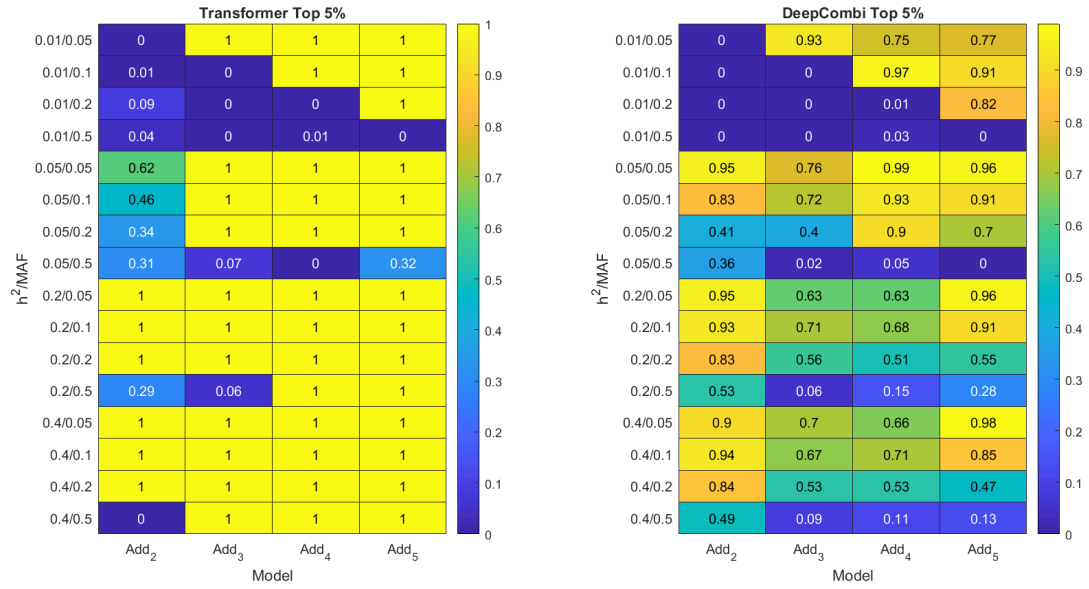

Figure 4.7: Additive Model Top 5%

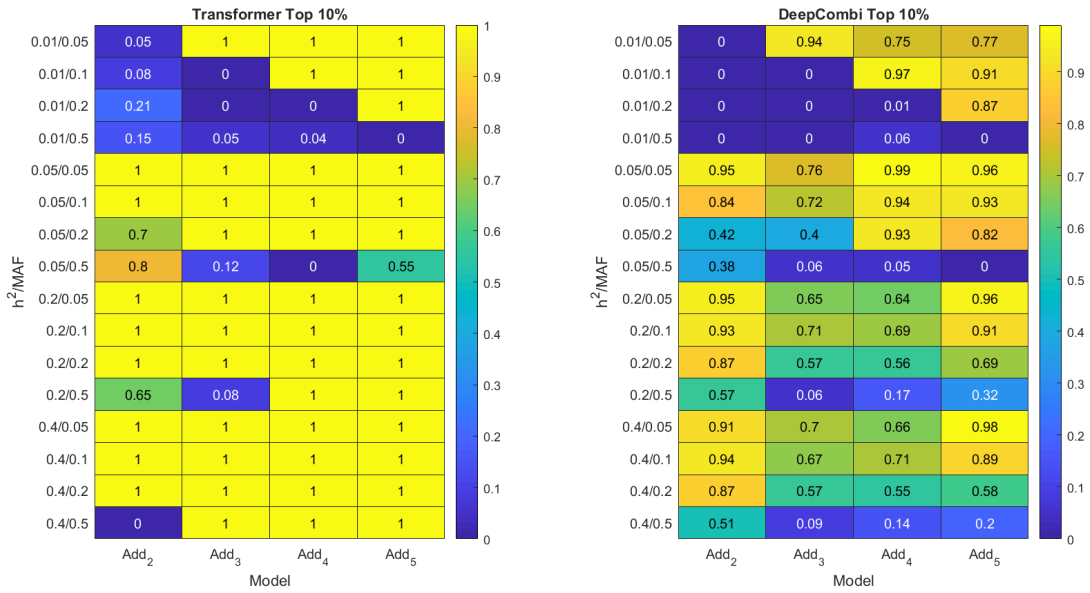

Figure 4.8: Additive Model Top 10%

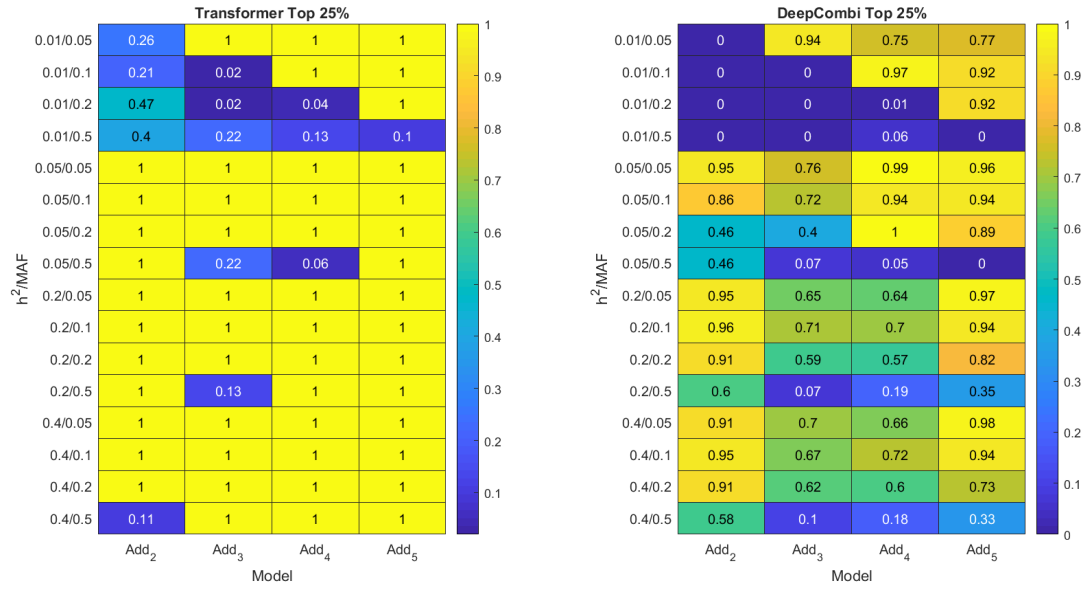

Figure 4.9: Additive Model Top 25%

A general overview, starting with top 5%, shows that DeepCOMBI finds the interacting SNPs in 54% of all datasets, while the transformer finds them in 72.8%, representing an improvement of almost 20% by comparison. For top 10%, DeepCOMBI finds interacting SNPs in 55.7%, while the transformer finds them in 77.3%, representing an improvement of 22%. Finally, for top 25%, DeepCOMBI finds interacting SNPs in 57.8%, while the transformer finds them in 81.8%, representing an improvement of 24%.

Focusing now on the top 5% (Figure 4.7), the results for different heritabilities are analyzed. As heritability controls the proportion of the population that exhibits observable differences due to genetic variance, it is expected that, for higher values of heritability, the transformer learns which SNPs may be interacting in a dataset with higher success.

For low heritabilities (h^2 = 0.01 and h^2 = 0.05), DeepCOMBI finds interacting SNPs in 47.1% of all datasets (3200), while the transformer finds them in 54%, an improvement of 7%. On the other hand, for high heritabilities (h^2 = 0.2 and h^2 = 0.4), DeepCOMBI finds interacting SNPs in 60.8% of all datasets (3200), while the transformer finds them in 91.7%, outperforming DeepCOMBI by 31%.

Regarding the explored interaction orders, DeepCOMBI finds interacting SNPs in 56%, 42.3%, 53.8%, and 63.8% of all datasets (1600 for each interaction order), considering second, third, fourth, and fifth order interactions, respectively. The same analysis for the transformer shows that interacting SNPs are found on 51%, 69.5%, 81.3%, and 89.5% of the considered datasets, which outperforms DeepCOMBI, but also suggests that, as the interaction order increases, the interacting SNPs are easier to find.

Figures 4.10 to 4.13 represent the average accuracy, precision, recall, and F1 score, respectively, for the transformer and DeepCOMBI. Given that all metrics have similar results, this analysis will focus only on accuracy. DeepCOMBI and the transformer exhibit very similar average accuracy values (72.74% and 72.71%, respectively).

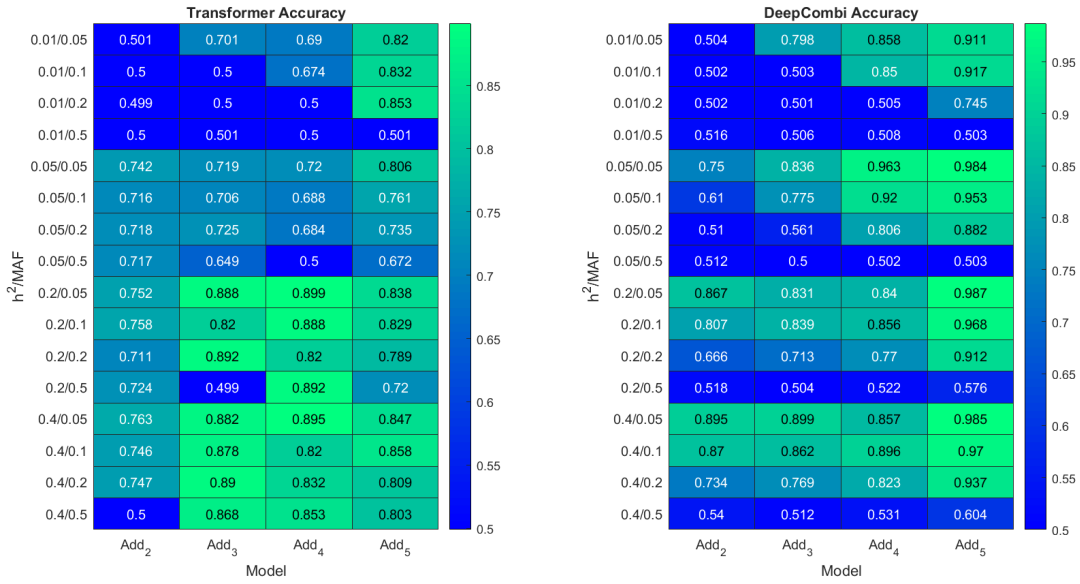

Figure 4.10: Additive Model Accuracy

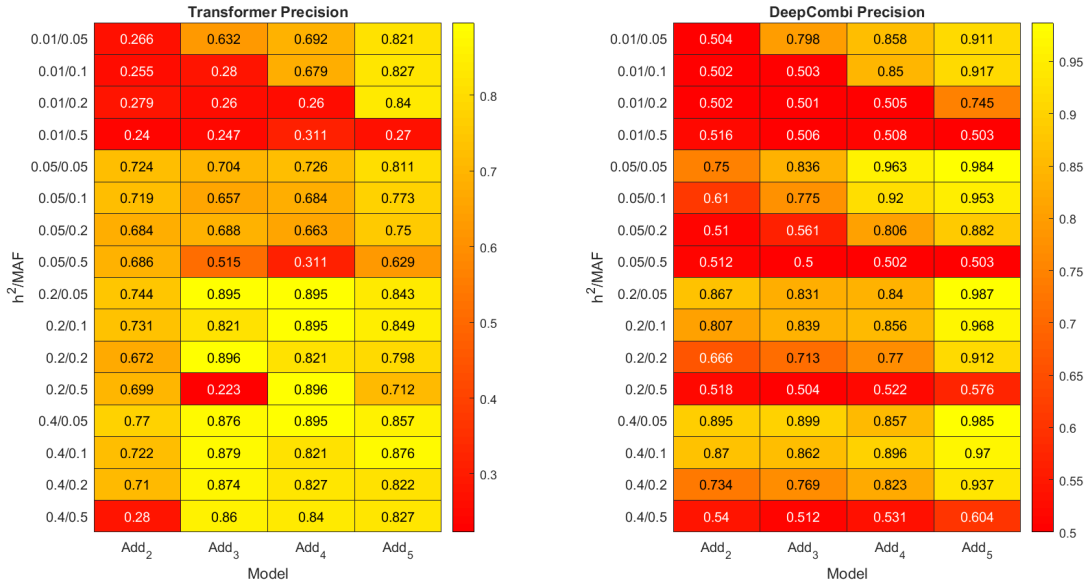

Figure 4.11: Additive Model Precision

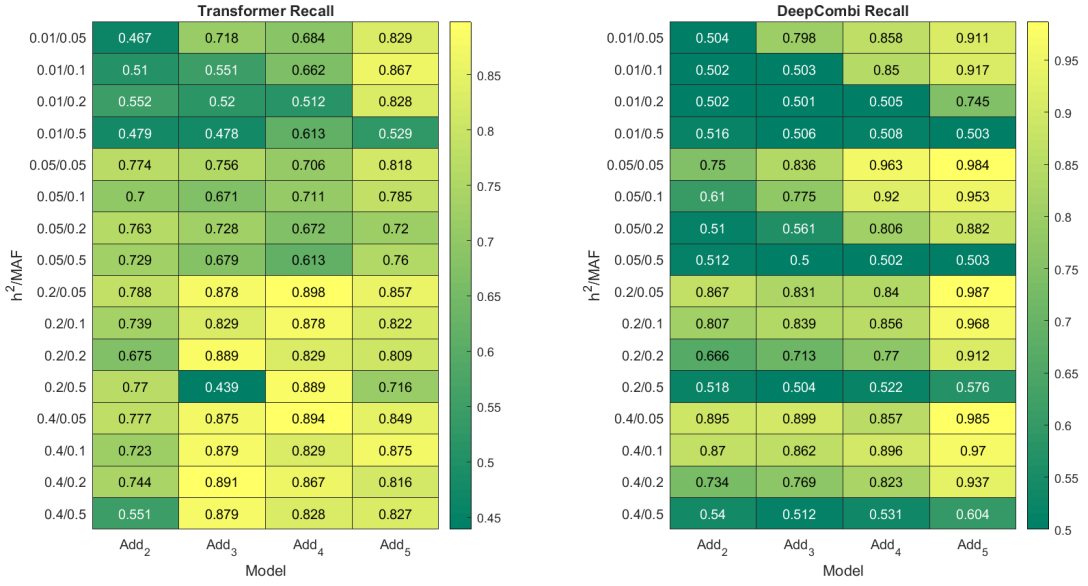

Figure 4.12: Additive Model Recall

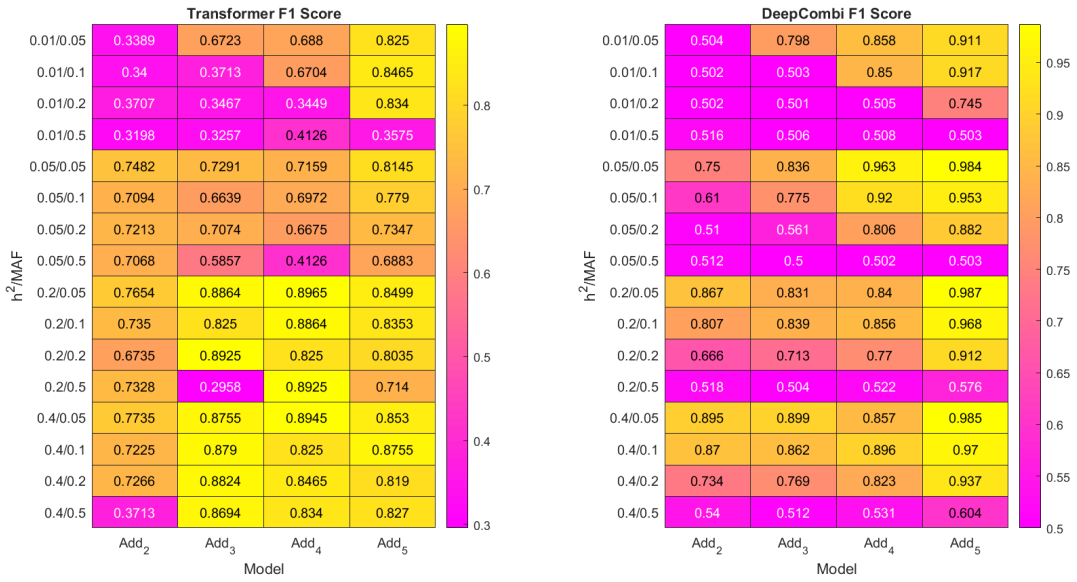

Figure 4.13: Additive Model F1 Score

Evaluating the results for low heritabilities (h^2 = 0.01 and h^2 = 0.05), DeepCOMBI achieves 67.8% accuracy, while the transformer achieves 65.1% accuracy. On the other hand, for high heritabilities (h^2 = 0.2 and h^2 = 0.4), DeepCOMBI achieves 77.6% accuracy, while the transformer achieves 80.3% accuracy. These values are correlated with the previous results, as identifying interacting SNPs on more datasets leads to higher accuracy values.

4.4.2 Multiplicative Models

For multiplicative models, Figure 4.14 shows the percentage of datasets where interacting SNPs had attention scores that were on the top 10% of all scores. The total number of training datasets is 5800, as there are four interaction orders, sixteen combinations of MAF and heritability, and 100 datasets for each combination, but for fifth order, six combinations are impossible to solve for penetrance tables.

A general overview shows that, for the top 10%, DeepCOMBI finds the interacting SNPs in 28.2% of all datasets, while the transformer finds them in 55.5%, representing an improvement of 27.3% by comparison. For low heritabilities (h^2 = 0.01 and h^2 = 0.05), DeepCOMBI finds interacting SNPs in 1.57% of all datasets (2800), while the transformer finds them in 9.03%, an improvement of 7.5%. On the other hand, for high heritabilities (h^2 = 0.2 and h^2 = 0.4), DeepCOMBI finds interacting SNPs in 53.1% of all datasets (3000), while the transformer finds them in 99%, outperforming DeepCOMBI by 46%.

Regarding the explored interaction orders, DeepCOMBI finds interacting SNPs in 22.9%, 36.7%, 29.9%, and 20.6% of all datasets (1600 for each interaction order, except fifth order, which has 1000

datasets), considering second, third, fourth, and fifth order interactions, respectively. The same analysis for the transformer shows that interacting SNPs are found on 61.3%, 52.4%, 50.3%, and 60% of the considered datasets, which outperforms DeepCOMBI.

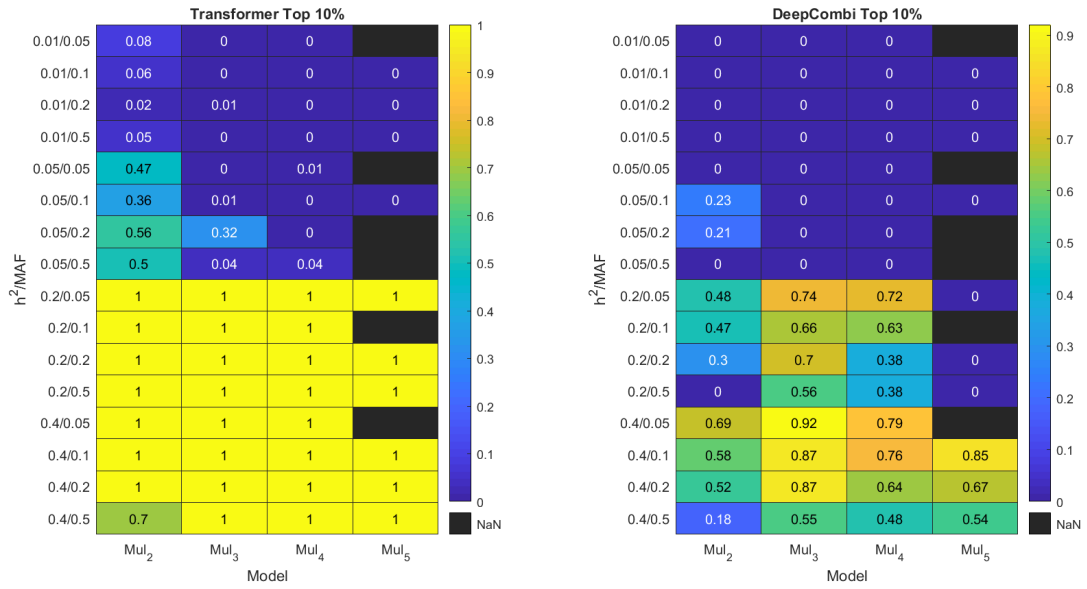

Figure 4.14: Multiplicative Model Top 10%

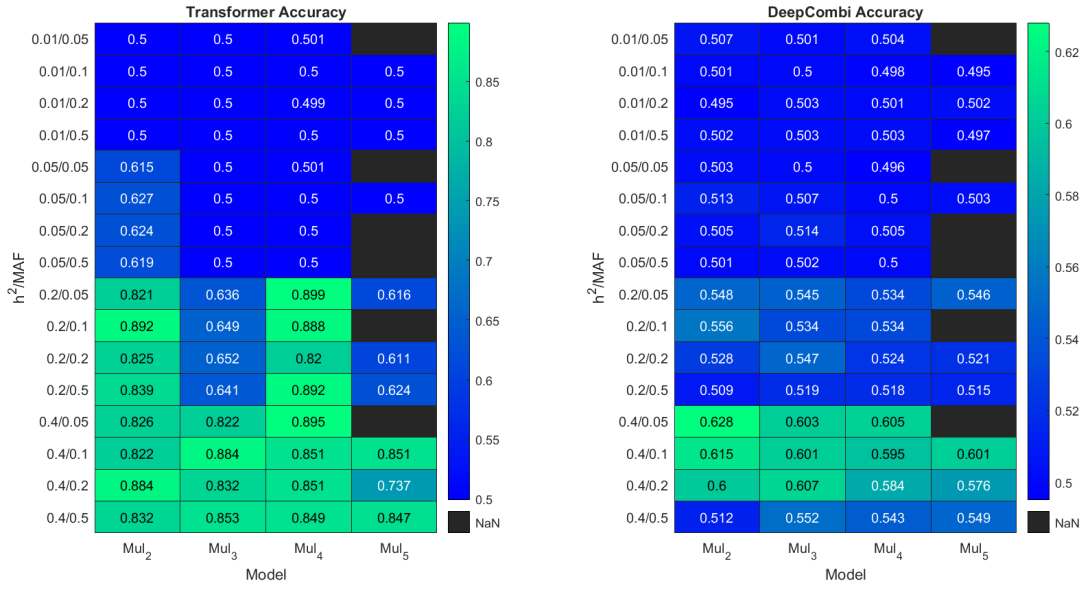

Figure 4.15: Multiplicative Model Accuracy

Figure 4.15 represents the average accuracy for the transformer and DeepCOMBI. DeepCOMBI

and the transformer exhibit average accuracy values of 53.1% and 66.2%, respectively. Evaluating the results for low heritabilities (h^2 = 0.01 and h^2 = 0.05), DeepCOMBI achieves 50.2% accuracy, while the transformer achieves 51.7% accuracy. On the other hand, for high heritabilities (h^2 = 0.2 and h^2 = 0.4), DeepCOMBI achieves 55.8% accuracy, while the transformer achieves 79.8% accuracy. These values are correlated with the previous results, as identifying interacting SNPs on more datasets leads to higher accuracy values.

4.4.3 Threshold Models

For threshold models, Figure 4.16 shows the percentage of datasets where interacting SNPs had attention scores that were on the top 25% of all scores. The total number of training datasets is 6400, as there are four interaction orders, sixteen combinations of MAF and heritability, and 100 datasets for each combination.

A general overview shows that, for the top 25%, DeepCOMBI finds the interacting SNPs in 50.4% of all datasets, while the transformer finds them in 58.8%, representing an improvement of 8.4% by comparison. For low heritabilities (h^2 = 0.01 and h^2 = 0.05), DeepCOMBI finds interacting SNPs in 16.5% of all datasets (3200), while the transformer finds them in 17.6%, an improvement of 1%. On the other hand, for high heritabilities (h^2 = 0.2 and h^2 = 0.4), DeepCOMBI finds interacting SNPs in 84.1% of all datasets (3200), while the transformer always finds them (100%), outperforming DeepCOMBI by 15.9%.

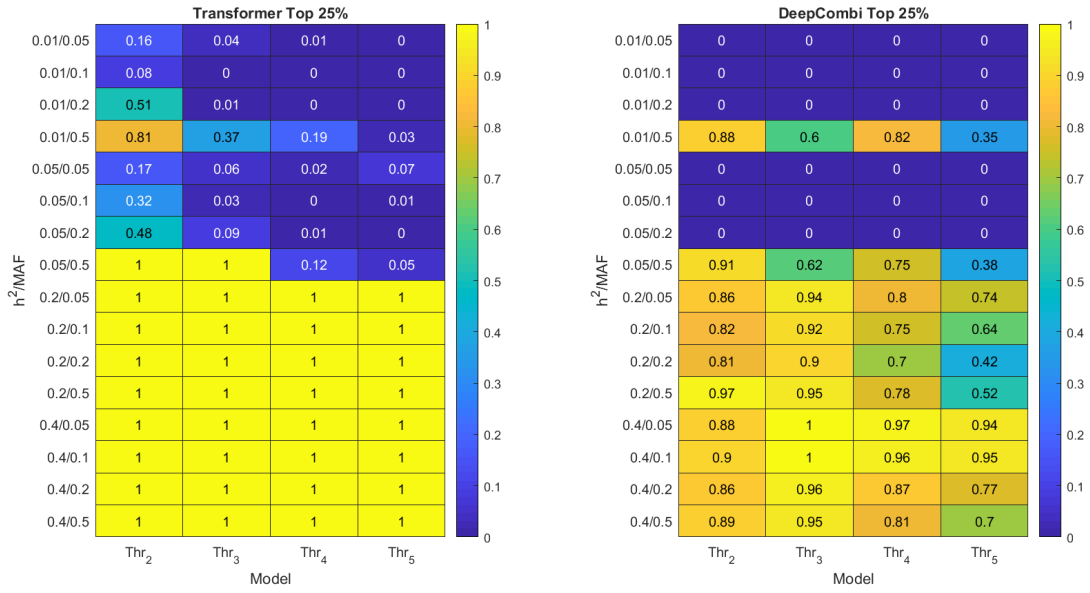

Figure 4.16: Threshold Model Top 25%

Regarding the explored interaction orders, DeepCOMBI finds interacting SNPs in 54.9%, 55.2%, 51.3%, and 40% of all datasets (1600 for each interaction order), considering second, third, fourth, and fifth order interactions, respectively. The same analysis for the transformer shows that interacting SNPs are found on 72.1%, 60%, 52.2%, and 51% of the considered datasets, which outperforms DeepCOMBI, but also suggests that, as the interaction order increases, the transformer finds interacting SNPs in less datasets.

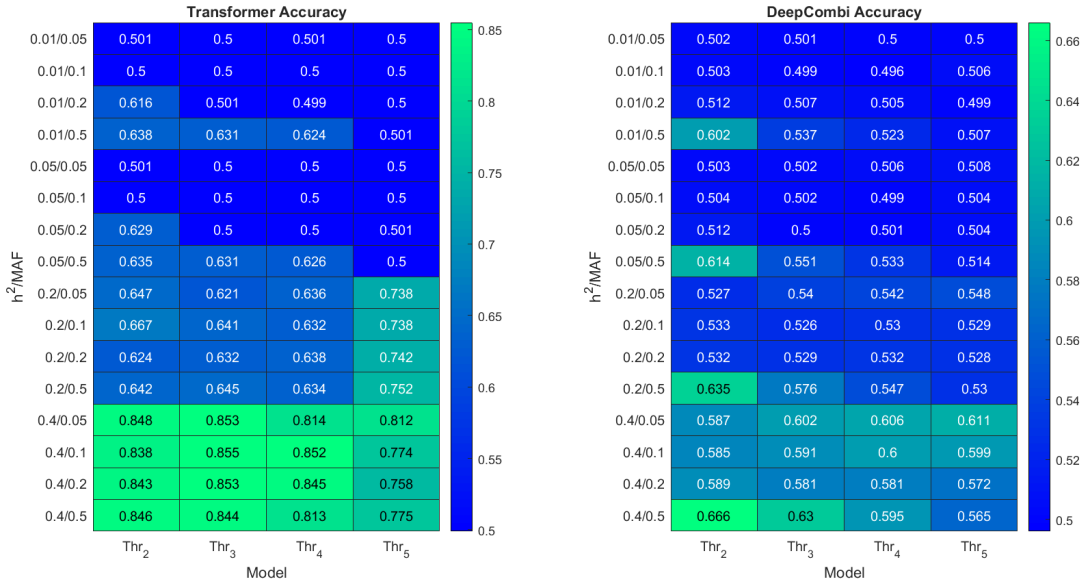

Figure 4.17: Threshold Model Accuracy

Figure 4.17 represents the average accuracy for the transformer and DeepCOMBI. DeepCOMBI and the transformer exhibit average accuracy values of 54.2% and 63.9%, respectively. Evaluating the results for low heritabilities (h^2 = 0.01 and h^2 = 0.05), DeepCOMBI achieves 51.4% accuracy, while the transformer achieves 53.2% accuracy. On the other hand, for high heritabilities (h^2 = 0.2 and h^2 = 0.4), DeepCOMBI achieves 57.6% accuracy, while the transformer achieves 74.5% accuracy. These values are correlated with the previous results, as identifying interacting SNPs on more datasets leads to higher accuracy values.

4.4.4 Xor Models

For xor models, Figure 4.18 shows the percentage of datasets where interacting SNPs had attention scores that were on the top 25% of all scores. The total number of training datasets is 6400, as there are four interaction orders, sixteen combinations of MAF and heritability, and 100 datasets for each combination.

A general overview shows that, for the top 25%, DeepCOMBI finds the interacting SNPs in 21% of all datasets, while the transformer finds them in 52.1%, representing an improvement of 31.1% by comparison. For low heritabilities (h^2 = 0.01 and h^2 = 0.05), DeepCOMBI finds interacting SNPs in 4.78% of all datasets (3200), while the transformer finds them in 17.3%, an improvement of almost 13%. On the other hand, for high heritabilities (h^2 = 0.2 and h^2 = 0.4), DeepCOMBI finds interacting SNPs in 37.2% of all datasets (3200), while the transformer finds them in 87%, outperforming DeepCOMBI by 50%.

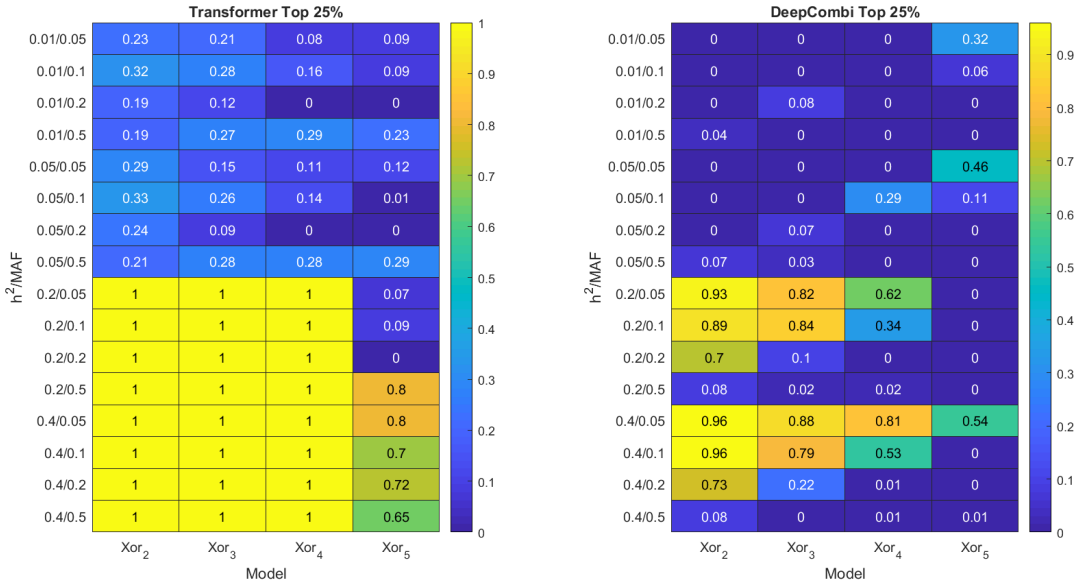

Figure 4.18: Xor Model Top 25%

Regarding the explored interaction orders, DeepCOMBI finds interacting SNPs in 34%, 24%, 16.4%, and 9.4% of all datasets (1600 for each interaction order), considering second, third, fourth, and fifth order interactions, respectively. The same analysis for the transformer shows that interacting SNPs are found on 62.5%, 60.4%, 56.6%, and 29.1% of the considered datasets, which outperforms DeepCOMBI, but also suggests that, as the interaction order increases, the transformer finds interacting SNPs in less datasets, similar to the results obtained for the threshold model.

Figure 4.19 represents the average accuracy for the transformer and DeepCOMBI. DeepCOMBI and the transformer exhibit average accuracy values of 51.5% and 62.2%, respectively. Evaluating the results for low heritabilities (h^2 = 0.01 and h^2 = 0.05), DeepCOMBI achieves 50.8% accuracy, while the transformer achieves 54.5% accuracy. On the other hand, for high heritabilities (h^2 = 0.2 and h^2 = 0.4), DeepCOMBI achieves 52.2% accuracy, while the transformer achieves 69.8% accuracy. These values are correlated with the previous results, as identifying interacting SNPs on more datasets leads to higher accuracy values.

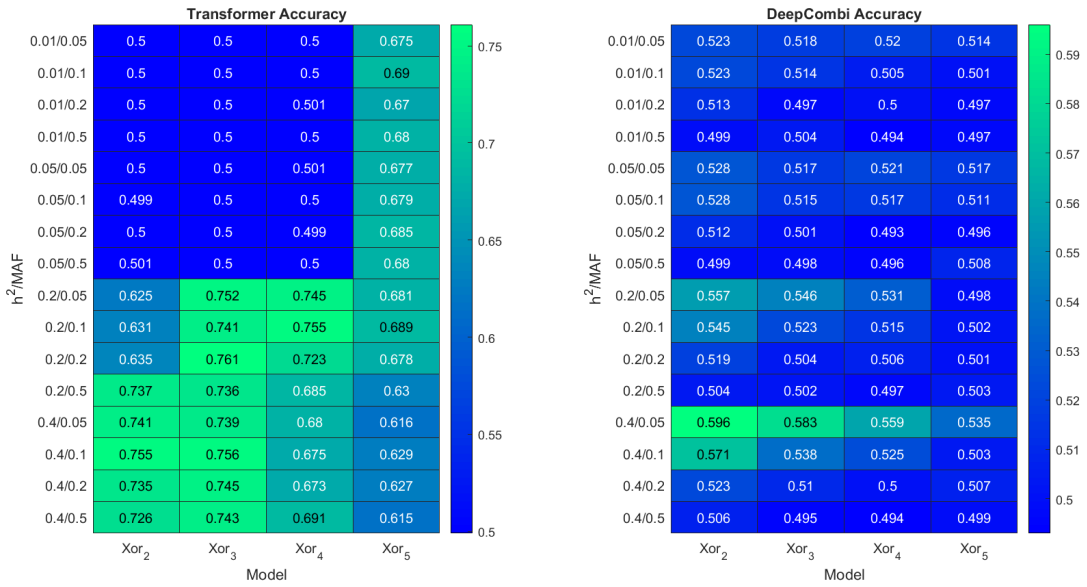

Figure 4.19: Xor Model Accuracy

4.5 Performance Evaluation on Hardware Platforms

So far, the experiments have shown the potential of transformers for epistasis detection, but have not yet shown the potential of Graphcore IPUs, a platform that is tailored for machine learning workloads. For this reason, it is necessary to understand how the performance of the proposed approach on IPU scales with the input datasets. Deep Learning models do not make assumptions on the dataset and interaction orders do not affect the algorithm's running time, as opposed, for example, to exhaustive search methods. Furthermore, the network architecture and number of epochs are kept fixed throughout the experiments. Therefore, the only parameters that can affect the neural network's running time on the Graphcore IPU are the number of patients (P) and the number of SNPs (N).

To evaluate and understand the capabilities of Graphcore IPUs when executing the proposed framework, its performance is evaluated and compared to NVIDIA A100 GPUs and Google TPUs V3. To this end, datasets targeting a range of patients from 400 to 6400 and a range of SNPs from 250 to 4000 are created to measure the influence of these parameters on the performance of the aforementioned platforms. For each combination of patient and SNP values, 100 datasets are trained on the transformer on all computer systems to measure the average training time. Tables 4.8 to 4.10 present the execution times for the NVIDIA A100 GPU (6912 CUDA cores, 80 GB memory), Google TPU V3 (2048 cores, 32 GB memory), and Graphcore IPU GC-200 (1472 tiles, 900 MB memory), respectively, for each combination of SNPs and patients.

Table 4.8: NVIDIA A100 GPU Computational Times for training the transformer on 100 datasets.

P \ N	250	500	1000	2000	4000
400	3m00s	3m41s	5m36s	8m36s	16m15s
800	5m47s	7m10s	11m02s	16m59s	32m18s
1600	11m17s	14m07s	20m58s	33m45s	1h04m22s
3200	22m27s	27m58s	40m29s	1h09m12s	2h08m57s
6400	44m30s	55m02s	1h20m44s	2h20m50s	4h17m39s

Table 4.9: Google TPU V3 (8 Cores) Computational Times for training the transformer on 100 datasets.

P \ N	250	500	1000	2000	4000
400	11m56s	12m01s	12m08s	12m23s	12m56s
800	15m20s	15m26s	15m39s	16m09s	16m50s
1600	21m56s	22m15s	22m45s	23m36s	24m57s
3200	35m16s	35m57s	36m45s	38m18s	41m08s
6400	1h02m09s	1h03m07s	1h04m49s	1h08m13s	1h14m14s

Table 4.10: IPU GC-200 Computational Times for training the transformer on 100 datasets.

P \ N	250	500	1000	2000	4000
400	8m43s	8m52s	9m38s	12m04s	15m20s
800	9m32s	10m00s	10m20s	14m04s	20m17s
1600	11m34s	12m21s	14m23s	20m19s	30m59s
3200	16m06s	17m45s	21m51s	31m58s	51m13s
6400	25m06s	28m48s	36m29s	55m36s	1h32m22s

Comparing the Graphcore IPU and NVIDIA GPU results, it is possible to verify that, for datasets with a small number of SNPs, the IPU provides lower performance. However, as the dataset size increases, both in SNPs and patients, the IPU scales better than the GPU, obtaining speedups higher than 2x for the largest datasets. The results for both platforms yield an interesting result, as it is possible to verify that the GPU execution times scale with the number of patients (whenever the number of patients double, the necessary time to train the network on all datasets also doubles), due to the resources being fully used. However, no such behavior is observed for the IPU. Profiling the IPU shows that, for small datasets, the resources are not fully used while it is training the network, therefore explaining why it does not scale in the same way as the GPU.

Comparing the IPU and TPU results, an interesting result to observe is that, as the number of SNPs increases, the IPU loses in computational time, when compared to the TPU. This is not unexpected, as

the number of SNPs directly influences the size of tensors used in the network's training operations, for which the TPU should be optimized. On the other hand, fixing the number of SNPs and increasing the number of patients seems to be detrimental to the TPU's performance. Similar to the IPU, the TPU does not seem to scale with the number of patients or with the number of SNPs in the same way as GPUs scale. This result may be a consequence of using multiple tensor cores to train the network in a parallel fashion, allowing for more computations in a time interval, and therefore reducing the necessary training time.

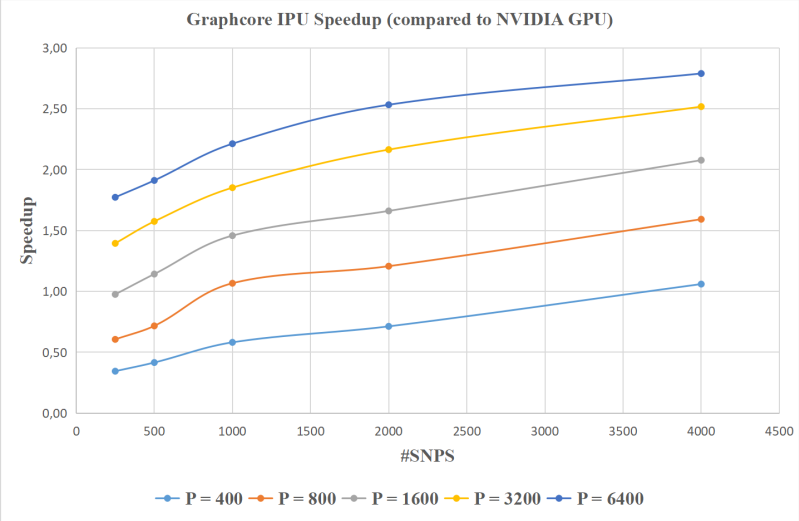

Figure 4.20: Graphcore IPU Speedup (compared to NVIDIA A100 GPU)

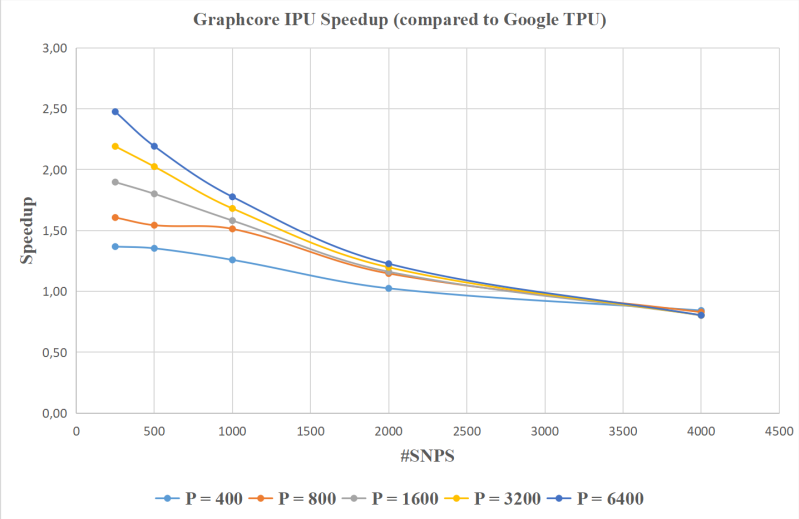

Figure 4.21: Graphcore IPU Speedup (compared to Google TPU V3)

Figures 4.20 and 4.21 depict the obtained speedups for the IPU when compared to the GPU and the TPU, respectively. For each figure, the X axis represents the number of SNPs (N), the Y axis

represents the speedups achieved with the IPU, and each line represents a fixed number of patients (P). Figure 4.20 shows that, for a fixed number of patients, IPUs achieve higher speedups than GPUs as the number of SNPs increases, validating the previous analysis. On the other hand, 4.21 shows the opposite behavior, as the IPU shows lower speedups, for a fixed number of patients and increasing number of SNPs, which also validates the previous analysis.

4.6 Application on a Real Dataset

In this section, the transformer is applied to a real breast cancer dataset [101] to evaluate whether attention scores capture the interacting SNPs and whether the network is trustworthy.

The aforementioned dataset has a total of 10000 samples, with 5000 cases and 5000 controls, and each sample has 23 SNPs. Previous studies on the same dataset [102] have found, using exhaustive search algorithms, interactions of order two (”rs2010204”, ”rs1007590”), three (”rs2010204”, ”rs1007590”, ”rs660049”), and four (”rs2010204”, ”rs0504248”, ”rs660049”, ”rs500760”). However, because the number of SNPs is small, it is possible to exhaustively search for higher order interactions. By using K2 score as an objective function, all combinations from second to eighth order were searched to find, for each interaction order, which was the SNP combination that best explained the observed phenotype. This search confirms the results from [102] for second to fourth order interactions, while also finding interactions of order five to order eight, as is presented in Table 4.11.

Table 4.11: Breast Cancer Dataset Epistatic Interactions

Breast Cancer Dataset Epistatic Interactions						
2nd Order	3rd Order	4th Order	5th Order	6th Order	7th Order	8th Order
rs2010204	rs2010204	rs2010204	rs2010204	rs2010204	rs2010204	rs2010204
rs1007590	rs1007590	rs0504248	rs0504248	rs0504248	rs0504248	rs0504248
	rs660049	rs660049	rs660049	rs660049	rs660049	rs660049
		rs500760	rs500760	rs500760	rs500760	rs500760
			rs00570070	rs00570070	rs00570070	rs00570070
				rs9240799	rs9240799	rs9240799
					rs521000	rs521000
						rs9478149

Running the transformer on the breast cancer dataset yields the results that are depicted in Figure 4.22, which displays the plot of attention scores for each SNP, sorted by their attention score. Considering the results from Table 4.11, it is possible to verify that SNPs ”rs2010204” and ”rs1007590”, which correspond to the second order interaction, are on the top 3 SNPs, with SNP ”rs2010204” having the highest attention score among all SNPs.

Furthermore, SNPs ”rs0504248”, ”rs660049”, and ”rs500760”, which are involved in third and fourth order interactions, have the second, sixth, and seventh highest attention scores, respectively. This implies that, within the first seven SNPs (top 30%), the transformer has already captured the predicted epistatic interactions from second to fourth order. Within the first nine SNPs (top 40%), the transformer also captures the fifth order interaction, as the only SNP missing (”rs00570070”) holds the ninth highest attention score. These results are depicted in Figure 4.23.

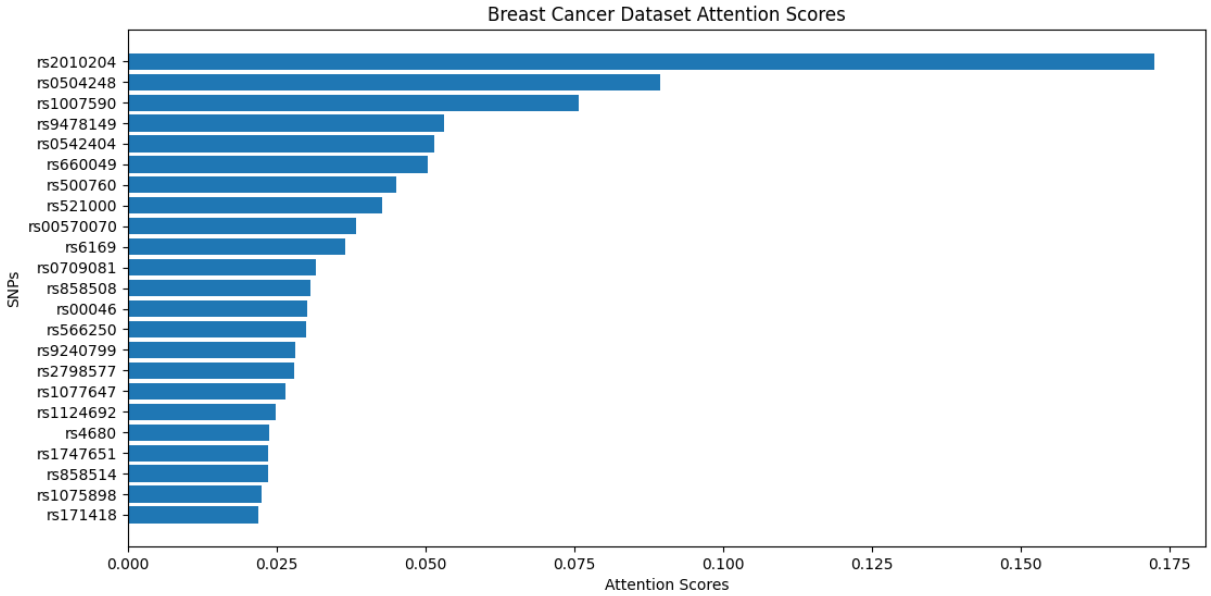

Figure 4.22: Breast Cancer Dataset Attention Scores. The SNPs are ordered decreasingly according to their attention scores.

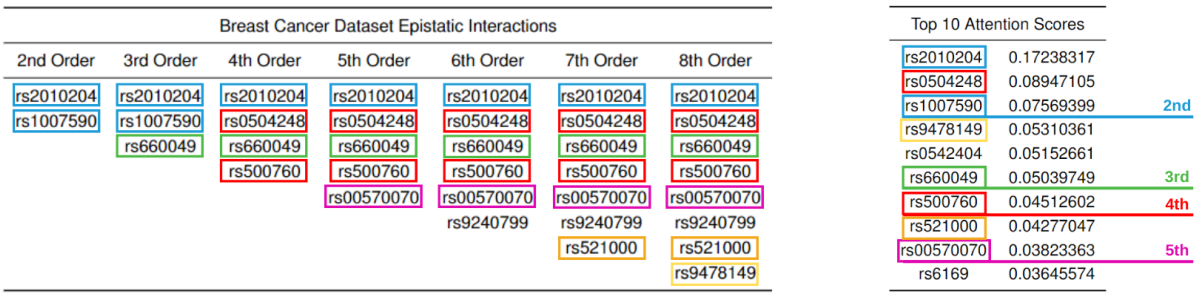

Figure 4.23: Breast Cancer Dataset Top 10 Attention Scores. Second to fifth order interactions are found within the top nine SNPs.

For sixth and higher order interactions, SNP ”rs9240799”, which is predicted in these interactions, has the fifteenth highest attention score, suggesting that the transformer does not identify it as a relevant SNP, compared to the previously analyzed interacting SNPs. Nevertheless, the presented results are promising, specially when the performance metrics are considered. As with the previous experiments,

this dataset was divided in two subsets, with 90% of the dataset being used for training and the remaining 10% used for testing. On the test subset, the transformer could predict the patients' phenotype with 100% accuracy, precision, recall, and F1 score. These results demonstrate the transformer's reliability, as it can learn epistatic interactions and use this knowledge to make accurate predictions on the phenotype of unknown patients.

4.7 Summary

In this section, the transformer's experimental results on multiple epistasis scenarios are presented and analyzed.

First, a hyperparameter optimization is performed, using a grid search strategy. To keep the number of possible networks from growing exponentially, the relevant hyperparameters to optimize are selected and their impact on the transformer's performance is studied. A more thorough analysis is performed for embedding algorithms by testing each one on all training datasets, since it is unknown which algorithm would be better for the problem of epistasis detection.

After finding the optimal network, the transformer is tested on simulated datasets that cover various epistasis models and compared to DeepCOMBI, a state-of-the-art technique that also employs neural networks and emphasizes network interpretability. The results show that the transformer is more capable of finding interacting SNPs in various epistasis models and interaction orders, proving it can be a reliable method to detect high order epistasis. Next, IPUs are compared with GPUs and TPUs in terms of training time. Running datasets with various values of SNPs and patients on all the platforms, the IPU achieves speedups above 2x in some cases, when compared to GPUs and TPUs, proving its potential to reduce the computational load of machine learning workloads.

To conclude the study of the transformer for epistasis detection, tests on a real breast cancer dataset are performed. Within the top 30% most relevant SNPs, the transformer captured second, third, and fourth order interactions, and within the top 40%, the transformer also identified a fifth order interaction. These results demonstrate the transformer's reliability in finding epistatic interactions.

Chapter 5

Conclusions and Future Work

Solving the problem of high order epistasis detection is essential to explain complex diseases. To tackle this issue, many different algorithms have been proposed in the state-of-the-art. The review of the existent approaches shows the potential of deep learning models to predict phenotypes from GWAS datasets, ignoring, however, the necessary biological interpretation to understand the underlying mechanisms of diseases. Therefore, the main objective of this dissertation is to fill the gap between prediction and interpretation by developing methodologies for network explainability that are not yet present in the state-of-the-art.

To fulfill this objective, a novel framework based on transformers, a neural network that has not yet been applied for epistasis detection, is proposed in the scope of this thesis. The capabilities for network explainability by means of the attention mechanism, coupled with its state-of-the-art results for NLP, demonstrate its potential for epistasis detection. Using dimensionality reduction techniques to find dense embeddings to represent SNPs, it is possible to run the transformer on epistasis datasets and evaluate the attention scores to find which are the most relevant SNPs to predict a patient's phenotype.

Given that deep learning approaches often have thousands of parameters and that the transformer does not scale well for long sequences, the implementation of this network is focused on sparsity, as sparse transformers are good enough to maintain performance and scale to long sequences. As the attention mechanism is the main bottleneck of the transformer, strategies to sparsify it are tested, such as Top-KAST, which zeroes the weights based on magnitude.

In the experimental work, after a hyperparameter optimization, the transformer was tested under a wide variety of epistasis models to compare the chosen algorithms for SNP embeddings. Afterwards, the results obtained for the optimal network were compared to other network explainability methods. This study shows that the transformer exhibits a better performance for epistasis detection, when compared to state-of-the-art techniques, for several epistasis models.

To demonstrate the Graphcore IPU's performance for machine learning, the transformer was tested

on this platform, on NVIDIA GPUs, and Google TPUs for datasets with varying number of samples and SNPs. The goal was to analyze the evolution of computational time with the input and compare all platforms. It is observed that the IPU achieves reasonable speedups when compared to GPUs and TPUs, showing that it can boost the training of sparse machine learning models for epistasis detection.

As a final step, the transformer was tested on a real breast cancer dataset to evaluate whether the network could be trusted. The results demonstrate that the transformer finds second to fourth order interactions on this dataset within the top 30% most relevant SNPs, and a fifth order interaction within the top 40%, while still predicting the phenotype with high accuracy. For this reason, the work developed in this dissertation could be valuable to solve the issue of the black-box nature of deep learning models.

5.1 Future Work

To overcome the black-box nature of deep learning models, a methodology based on transformers was proposed. However, as the transformer has never been used before to tackle the epistasis detection problem, there are aspects of the developed framework that should be further analyzed and solved to improve its performance.

In this dissertation, the number of samples, SNPs, and case-control ratio are kept fixed. Analyzing the impact of these parameters on the transformer's performance is important, given the novelty of this technique for epistasis detection. Furthermore, while interactions up to fifth order were tested for this framework, it is possible, for some epistasis models, to tackle even higher interaction orders. Therefore, the transformer should also be tested on datasets that simulate sixth order epistasis and higher. Additionally, datasets that do not exhibit marginal effects should also be considered to further test the performance of the proposed framework.

To obtain SNP embeddings, dimensionality reduction techniques were used, but there may be better techniques to map SNPs that can further improve the transformer's performance. Regarding the network's hyperparameter optimization, a wider exploration of the hyperparameters should be made, as many were kept fixed in the network to avoid a combinatorial explosion. As some of the proposed embeddings use K-Nearest Neighbors, the value of K may also be an interesting hyperparameter to analyze. Finally, sparse transformers should be further explored in the future to take better advantage of the IPU's architecture. Furthermore, for future analysis, the IPU should not only be compared with other platforms in terms of computational time, but also in terms of energy efficiency.

Bibliography

[1] P. M. Visscher, N. R. Wray, Q. Zhang, P. Sklar, M. I. McCarthy, M. A. Brown, and J. Yang, "10 Years of GWAS Discovery: Biology, Function, and Translation," *The American Journal of Human Genetics*, vol. 101, no. 1, pp. 5–22, Jul. 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0002929717302409

[2] K. Hemminki, A. Försti, and J. L. Bermejo, "The 'common disease-common variant'hypothesis and familial risks," *PloS one*, vol. 3, no. 6, p. e2504, 2008.

[3] R. J. Klein, C. Zeiss, E. Y. Chew, J.-Y. Tsai, R. S. Sackler, C. Haynes, A. K. Henning, J. P. SanGiovanni, S. M. Mane, S. T. Mayne, M. B. Bracken, F. L. Ferris, J. Ott, C. Barnstable, and J. Hoh, "Complement factor H polymorphism in age-related macular degeneration," *Science (New York, N.Y.)*, vol. 308, no. 5720, pp. 385–389, Apr. 2005, edition: 2005/03/10. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/15761122

[4] A. V. Khera, M. Chaffin, K. H. Wade, S. Zahid, J. Brancale, R. Xia, M. Distefano, O. Senol-Cosar, M. E. Haas, A. Bick, K. G. Aragam, E. S. Lander, G. D. Smith, H. Mason-Suares, M. Fornage, M. Lebo, N. J. Timpson, L. M. Kaplan, and S. Kathiresan, "Polygenic Prediction of Weight and Obesity Trajectories from Birth to Adulthood," *Cell*, vol. 177, no. 3, pp. 587–596.e9, Apr. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0092867419302909

[5] C. Niel, C. Sinoquet, C. Dina, and G. Rocheleau, "A survey about methods dedicated to epistasis detection," *Frontiers in Genetics*, vol. 6, p. 285, 2015. [Online]. Available: https://www.frontiersin.org/article/10.3389/fgene.2015.00285

[6] B. Maher, "Personal genomes: The case of the missing heritability," *Nature*, vol. 456, no. 7218, pp. 18–21, Nov. 2008. [Online]. Available: https://doi.org/10.1038/456018a

[7] N. Perdigones, A. G. Vigo, J. R. Lamas, A. Martínez, A. Balsa, D. Pascual-Salcedo, E. G. de la Concha, B. Fernández-Gutiérrez, and E. Urcelay, "Evidence of epistasis between TNFRSF14 and TNFRSF6B polymorphisms in patients with rheumatoid arthritis," *Arthritis & Rheumatism*,

vol. 62, no. 3, pp. 705–710, Mar. 2010, publisher: John Wiley & Sons, Ltd. [Online]. Available: https://doi.org/10.1002/art.27292

[8] J. C. Turton, J. Bullock, C. Medway, H. Shi, K. Brown, O. Belbin, N. Kalsheker, M. M. Carrasquillo, D. W. Dickson, N. R. Graff-Radford, R. C. Petersen, S. G. Younkin, and K. Morgan, "Investigating Statistical Epistasis in Complex Disorders," *Journal of Alzheimer's Disease*, vol. 25, no. 4, pp. 635–644, 2011, publisher: IOS Press.

[9] T. LaFramboise, "Single nucleotide polymorphism arrays: a decade of biological, computational and technological advances," *Nucleic Acids Research*, vol. 37, no. 13, pp. 4181–4193, Jul. 2009. [Online]. Available: https://doi.org/10.1093/nar/gkp552

[10] T. F. Mackay and J. H. Moore, "Why epistasis is important for tackling complex human disease genetics," *Genome Medicine*, vol. 6, no. 6, p. 42, Jun. 2014. [Online]. Available: https://doi.org/10.1186/gm561

[11] L. S. Yung, C. Yang, X. Wan, and W. Yu, "GBOOST: a GPU-based tool for detecting gene–gene interactions in genome–wide case control studies," *Bioinformatics*, vol. 27, no. 9, pp. 1309–1310, May 2011.

[12] T. Kam-Thong, B. Pütz, N. Karbalai, B. Müller−Myhsok, and K. Borgwardt, "Epistasis detection on quantitative phenotypes by exhaustive enumeration using GPUs," *Bioinformatics*, vol. 27, no. 13, pp. i214–i221, Jul. 2011. [Online]. Available: https://doi.org/10.1093/bioinformatics/btr218

[13] R. Nobre, A. Ilic, S. Santander-Jiménez, and L. Sousa, "Exploring the Binary Precision Capabilities of Tensor Cores for Epistasis Detection," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2020, pp. 338–347, journal Abbreviation: 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS).

[14] R. Nobre, S. Santander-Jiménez, L. Sousa, and A. Ilic, "Accelerating 3-Way Epistasis Detection with CPU+GPU Processing," in *Job Scheduling Strategies for Parallel Processing*, D. Klusáček, W. Cirne, and N. Desai, Eds. Cham: Springer International Publishing, 2020, pp. 106–126.

[15] C. Ponte-Fernández, J. González-Domínguez, and M. J. Martín, "Fast search of third-order epistatic interactions on CPU and GPU clusters," *The International Journal of High Performance Computing Applications*, vol. 34, no. 1, pp. 20–29, Jan. 2020, publisher: SAGE Publications Ltd STM. [Online]. Available: https://doi.org/10.1177/1094342019852128

[16] R. Nobre, A. Ilic, S. Santander-Jiménez, and L. Sousa, "Retargeting Tensor Accelerators for Epistasis Detection," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 9, pp. 2160–2174, Sep. 2021.

[17] R. Nobre, A. Ilic, S. Santander-Jiménez, and L. Sousa, “Fourth-Order Exhaustive Epistasis Detection for the XPU Era,” in *50th International Conference on Parallel Processing*. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3472456.3472509

[18] L. Wienbrandt, J. C. Kässens, J. González-Domínguez, B. Schmidt, D. Ellinghaus, and M. Schimmler, “FPGA-based Acceleration of Detecting Statistical Epistasis in GWAS,” *2014 International Conference on Computational Science*, vol. 29, pp. 220–230, Jan. 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050914001975

[19] J. C. Kässens, L. Wienbrandt, J. González-Domínguez, B. Schmidt, and M. Schimmler, “High-speed exhaustive 3-locus interaction epistasis analysis on FPGAs,” *Computational Science at the Gates of Nature*, vol. 9, pp. 131–136, Jul. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S187775031500068X

[20] G. Ribeiro, N. Neves, S. Santander-Jiménez, and A. Ilic, “HEDAcc: FPGA-based Accelerator for High-order Epistasis Detection,” in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2021, pp. 124–132, journal Abbreviation: 2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM).

[21] M. Pérez-Enciso and L. M. Zingaretti, “A Guide on Deep Learning for Complex Trait Genomic Prediction,” *Genes*, vol. 10, no. 7, 2019.

[22] X.-W. Chen and X. Lin, “Big data deep learning: Challenges and perspectives,” *IEEE Access*, vol. 2, pp. 514–525, 2014.

[23] Y. Li, C. Huang, L. Ding, Z. Li, Y. Pan, and X. Gao, “Deep learning in bioinformatics: Introduction, application, and perspective in the big data era,” *Methods*, vol. 166, pp. 4–21, 2019.

[24] R. Ma and L. Niu, “A Survey of Sparse-Learning Methods for Deep Neural Networks,” in *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, 2018, pp. 647–650.

[25] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, “Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks,” 2021.

[26] I. Chelombiev, D. Justus, D. Orr, A. Dietrich, F. Gressmann, A. Koliousis, and C. Luschi, “GroupBERT: Enhanced Transformer Architecture with Efficient Grouped Structures,” 2021, _eprint: 2106.05822.

[27] S. Maddrell-Mander, L. R. M. Mohan, A. Marshall, D. O’Hanlon, K. Petridis, J. Rademacker, V. Rege, and A. Titterton, “Studying the Potential of Graphcore® IPUs for Applications in Particle

Physics," *Computing and Software for Big Science*, vol. 5, no. 1, p. 8, Mar. 2021. [Online]. Available: https://doi.org/10.1007/s41781-021-00057-z

[28] D. Masters, A. Labatie, Z. Eaton-Rosen, and C. Luschi, "Making EfficientNet More Efficient: Exploring Batch-Independent Normalization, Group Convolutions and Reduced Resolution Training," 2021, _eprint: 2106.03640.

[29] Z. Jia, B. Tillman, M. Maggioni, and D. P. Scarpazza, "Dissecting the Graphcore IPU Architecture via Microbenchmarking," 2019, _eprint: 1912.03413.

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, \. Kaiser, and I. Polosukhin, "Attention is All You Need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, pp. 6000–6010, event-place: Long Beach, California, USA.

[31] J. Piriyapongsa, C. Ngamphiw, A. Intarapanich, S. Kulawonganunchai, A. Assawamakin, C. Bootchai, P. J. Shaw, and S. Tongsima, "iLOCi: a SNP interaction prioritization technique for detecting epistasis in genome-wide association studies," *BMC Genomics*, vol. 13, no. 7, p. S2, Dec. 2012. [Online]. Available: https://doi.org/10.1186/1471-2164-13-S7-S2

[32] N. Risch and K. Merikangas, "The future of genetic studies of complex human diseases," *Science*, vol. 273, no. 5281, pp. 1516–1517, 1996.

[33] W. Bateson, "Mendel's Principles of Heredity," *Nature*, vol. 86, no. 2169, p. 407, 1911. [Online]. Available: https://doi.org/10.1038/086407a0

[34] P. Moran and C. Smith, "The correlation between relatives on the supposition of mendelian inheritance," *Transactions of the Royal Society of Edinburgh*, vol. 52, pp. 899–438, 1918.

[35] R. J. Urbanowicz, J. Kiralis, N. A. Sinnott-Armstrong, T. Heberling, J. M. Fisher, and J. H. Moore, "GAMETES: a fast, direct algorithm for generating pure, strict, epistatic models with random architectures," *BioData Mining*, vol. 5, no. 1, p. 16, Oct. 2012. [Online]. Available: https://doi.org/10.1186/1756-0381-5-16

[36] C. Ponte-Fernández, J. González-Domínguez, A. Carvajal-Rodríguez, and M. J. Martín, "Toxo: a library for calculating penetrance tables of high-order epistasis models," *BMC bioinformatics*, vol. 21, no. 1, pp. 1–9, 2020.

[37] B. González-Seoane, C. Ponte-Fernández, J. González-Domínguez, and M. J. Martín, "Pytoxo: a python tool for calculating penetrance tables of high-order epistasis models," *BMC bioinformatics*, vol. 23, no. 1, pp. 1–13, 2022.

[38] O. Mayo, “A century of hardy–weinberg equilibrium,” *Twin Research and Human Genetics*, vol. 11, no. 3, pp. 249–256, 2008.

[39] C. Ponte-Fernandez, J. Gonzalez-Dominguez, A. Carvajal-Rodriguez, and M. J. Martin, “Evaluation of Existing Methods for High-Order Epistasis Detection,” *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pp. 1–1, 2020.

[40] X. Wan, C. Yang, Q. Yang, H. Xue, X. Fan, N. L. S. Tang, and W. Yu, “BOOST: A Fast Approach to Detecting Gene-Gene Interactions in Genome-wide Case-Control Studies,” *The American Journal of Human Genetics*, vol. 87, no. 3, pp. 325–340, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0002929710003782

[41] Q. Wang, F. Shi, A. Kowalczyk, R. M. Campbell, B. Goudey, D. Rawlinson, A. Harwood, H. Ferra, and A. Kowalczyk, “Gwis fi: A universal gpu interface for exhaustive search of pairwise interactions in case-control gwas in minutes,” in *2014 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2014, pp. 403–409.

[42] M. D. Ritchie, “Finding the Epistasis Needles in the Genome-Wide Haystack,” in *Epistasis: Methods and Protocols*, J. H. Moore and S. M. Williams, Eds. New York, NY: Springer New York, 2015, pp. 19–33. [Online]. Available: https://doi.org/10.1007/978-1-4939-2155-3_2

[43] J. Liu, G. Yu, Y. Jiang, and J. Wang, “HiSeeker: Detecting High-Order SNP Interactions Based on Pairwise SNP Combinations,” *Genes*, vol. 8, no. 6, 2017.

[44] X. Zhang, F. Zou, and W. Wang, “FastChi: an efficient algorithm for analyzing gene-gene interactions,” *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, pp. 528–539, 2009. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/19209728

[45] S. Bhattacharjee, Z. Wang, J. Ciampa, P. Kraft, S. Chanock, K. Yu, and N. Chatterjee, “Using Principal Components of Genetic Variation for Robust and Powerful Detection of Gene-Gene Interactions in Case-Control and Case-Only Studies,” *The American Journal of Human Genetics*, vol. 86, no. 3, pp. 331–342, Mar. 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0002929710000303

[46] M. D. Ritchie, L. W. Hahn, N. Roodi, L. R. Bailey, W. D. Dupont, F. F. Parl, and J. H. Moore, “Multifactor-Dimensionality Reduction Reveals High-Order Interactions among Estrogen-Metabolism Genes in Sporadic Breast Cancer,” *The American Journal of Human Genetics*, vol. 69, no. 1, pp. 138–147, Jul. 2001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0002929707614530

[47] C. S. Greene, N. A. Sinnott-Armstrong, D. S. Himmelstein, P. J. Park, J. H. Moore, and B. T. Harris, "Multifactor dimensionality reduction for graphics processing units enables genome-wide testing of epistasis in sporadic ALS," *Bioinformatics*, vol. 26, no. 5, pp. 694–695, Mar. 2010. [Online]. Available: https://doi.org/10.1093/bioinformatics/btq009

[48] D. Gola, J. M. Mahachie John, K. van Steen, and I. R. König, "A roadmap to multifactor dimensionality reduction methods," *Briefings in Bioinformatics*, vol. 17, no. 2, pp. 293–308, Mar. 2016. [Online]. Available: https://doi.org/10.1093/bib/bbv038

[49] K. Kira, L. A. Rendell *et al.*, "The feature selection problem: Traditional methods and a new algorithm," in *Aaai*, vol. 2, no. 1992a, 1992, pp. 129–134.

[50] R. J. Urbanowicz, M. Meeker, W. La Cava, R. S. Olson, and J. H. Moore, "Relief-based feature selection: Introduction and review," *Journal of Biomedical Informatics*, vol. 85, pp. 189–203, Sep. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1532046418301400

[51] I. Kononenko, "Estimating attributes: Analysis and extensions of relief," in *European conference on machine learning*. Springer, 1994, pp. 171–182.

[52] B. Draper, C. Kaito, and J. Bins, "Iterative relief," in *2003 Conference on Computer Vision and Pattern Recognition Workshop*, vol. 6, 2003, pp. 62–62.

[53] Y. Sun and J. Li, "Iterative relief for feature weighting," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 913–920. [Online]. Available: https://doi.org/10.1145/1143844.1143959

[54] C. S. Greene, D. S. Himmelstein, J. Kiralis, and J. H. Moore, "The Informative Extremes: Using Both Nearest and Farthest Individuals Can Improve Relief Algorithms in the Domain of Human Genetics," in *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*, C. Pizzuti, M. D. Ritchie, and M. Giacobini, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 182–193.

[55] M. E. Stokes and S. Visweswaran, "Application of a spatially-weighted relief algorithm for ranking genetic predictors of disease," *BioData mining*, vol. 5, no. 1, pp. 1–11, 2012.

[56] D. Granizo-Mackenzie and J. H. Moore, "Multiple threshold spatially uniform relieff for the genetic analysis of complex human diseases," in *European conference on evolutionary computation, machine learning and data mining in bioinformatics*. Springer, 2013, pp. 1–10.

[57] R. J. Urbanowicz, R. S. Olson, P. Schmitt, M. Meeker, and J. H. Moore, "Benchmarking relief-based feature selection methods for bioinformatics data mining," *Journal of Biomedical*

Informatics, vol. 85, pp. 168–188, Sep. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1532046418301412

[58] C. S. Greene, N. M. Penrod, J. Kiralis, and J. H. Moore, "Spatially Uniform ReliefF (SURF) for computationally-efficient filtering of gene-gene interactions," *BioData Mining*, vol. 2, no. 1, p. 5, Sep. 2009. [Online]. Available: https://doi.org/10.1186/1756-0381-2-5

[59] J. Li, J. D. Malley, A. S. Andrew, M. R. Karagas, and J. H. Moore, "Detecting gene-gene interactions using a permutation-based random forest method," *BioData Mining*, vol. 9, no. 1, p. 14, Apr. 2016. [Online]. Available: https://doi.org/10.1186/s13040-016-0093-5

[60] R. Jiang, W. Tang, X. Wu, and W. Fu, "A random forest approach to the detection of epistatic interactions in case-control studies," *BMC Bioinformatics*, vol. 10, no. 1, p. S65, Jan. 2009. [Online]. Available: https://doi.org/10.1186/1471-2105-10-S1-S65

[61] L. E. Raileanu and K. Stoffel, "Theoretical comparison between the gini index and information gain criteria," *Annals of Mathematics and Artificial Intelligence*, vol. 41, no. 1, pp. 77–93, 2004.

[62] X. Li, "A fast and exhaustive method for heterogeneity and epistasis analysis based on multi-objective optimization," *Bioinformatics*, vol. 33, no. 18, pp. 2829–2836, Sep. 2017. [Online]. Available: https://doi.org/10.1093/bioinformatics/btx339

[63] P.-J. Jing and H.-B. Shen, "MACOED: a multi-objective ant colony optimization algorithm for SNP epistasis detection in genome-wide association studies," *Bioinformatics*, vol. 31, no. 5, pp. 634–641, Mar. 2015. [Online]. Available: https://doi.org/10.1093/bioinformatics/btu702

[64] Y. Chen, F. Xu, C. Pian, M. Xu, L. Kong, J. Fang, Z. Li, and L. Zhang, "EpiMOGA: An Epistasis Detection Method Based on a Multi-Objective Genetic Algorithm," *Genes*, vol. 12, no. 2, 2021.

[65] Y. Sun, J. Shang, J.-X. Liu, S. Li, and C.-H. Zheng, "epiACO - a method for identifying epistasis based on ant Colony optimization algorithm," *BioData Mining*, vol. 10, no. 1, p. 23, Jul. 2017. [Online]. Available: https://doi.org/10.1186/s13040-017-0143-7

[66] X. Lei, F. Wang, F.-X. Wu, A. Zhang, and W. Pedrycz, "Protein complex identification through markov clustering with firefly algorithm on dynamic protein–protein interaction networks," *Information Sciences*, vol. 329, pp. 303–316, 2016.

[67] X.-S. Yang, *Nature-inspired algorithms and applied optimization*. Springer, 2017, vol. 744.

[68] Y. Guo, Z. Zhong, C. Yang, J. Hu, Y. Jiang, Z. Liang, H. Gao, and J. Liu, "Epi-GTBN: an approach of epistasis mining based on genetic Tabu algorithm and Bayesian network," *BMC Bioinformatics*, vol. 20, no. 1, p. 444, Aug. 2019. [Online]. Available: https://doi.org/10.1186/s12859-019-3022-z

[69] L. Yuan, C.-A. Yuan, and D.-S. Huang, “FAACOSE: A Fast Adaptive Ant Colony Optimization Algorithm for Detecting SNP Epistasis,” *Complexity*, vol. 2017, p. 5024867, Sep. 2017, publisher: Hindawi. [Online]. Available: https://doi.org/10.1155/2017/5024867

[70] S. Uppu, A. Krishna, and R. P. Gopalan, “A deep learning approach to detect SNP interactions.” *J. Softw.*, vol. 11, no. 10, pp. 965–975, 2016.

[71] S. Uppu and A. Krishna, “An Intensive Search for Higher-Order Gene-Gene Interactions by Improving Deep Learning Model,” in *2018 IEEE 18th International Conference on Bioinformatics and Bioengineering (BIBE)*, 2018, pp. 104–109.

[72] X. Li, L. Liu, J. Zhou, and C. Wang, “Heterogeneity Analysis and Diagnosis of Complex Diseases Based on Deep Learning Method,” *Scientific Reports*, vol. 8, no. 1, p. 6155, Apr. 2018. [Online]. Available: https://doi.org/10.1038/s41598-018-24588-5

[73] A. Aghazadeh, H. Nisonoff, O. Ocal, D. H. Brookes, Y. Huang, O. O. Koyluoglu, J. Listgarten, and K. Ramchandran, “Epistatic Net allows the sparse spectral regularization of deep neural networks for inferring fitness functions,” *Nature Communications*, vol. 12, no. 1, p. 5225, Sep. 2021. [Online]. Available: https://doi.org/10.1038/s41467-021-25371-3

[74] C. A. C. Montañez, P. Fergus, C. Chalmers, N. H. A. H. Malim, B. Abdulaimma, D. Reilly, and F. Falciani, “SAERMA: Stacked Autoencoder Rule Mining Algorithm for the Interpretation of Epistatic Interactions in GWAS for Extreme Obesity,” *IEEE Access*, vol. 8, pp. 112 379–112 392, 2020.

[75] S. Salesi, A. A. Alani, and G. Cosma, “A Hybrid Model for Classification of Biomedical Data Using Feature Filtering and a Convolutional Neural Network,” in *2018 Fifth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, 2018, pp. 226–232.

[76] H. Wang, T. Yue, J. Yang, W. Wu, and E. P. Xing, “Deep mixed model for marginal epistasis detection and population stratification correction in genome-wide association studies,” *BMC Bioinformatics*, vol. 20, no. 23, p. 656, Dec. 2019. [Online]. Available: https://doi.org/10.1186/s12859-019-3300-9

[77] A. L. Beam, A. Motsinger-Reif, and J. Doyle, “Bayesian neural networks for detecting epistasis in genetic association studies,” *BMC Bioinformatics*, vol. 15, no. 1, p. 368, Nov. 2014. [Online]. Available: https://doi.org/10.1186/s12859-014-0368-0

[78] L. S. Glória, C. D. Cruz, R. A. M. Vieira, M. D. V. de Resende, P. S. Lopes, O. H. D. de Siqueira, and F. Fonseca e Silva, “Accessing marker effects and heritability estimates from genome prediction by Bayesian regularized neural networks,” *Livestock Science*, vol. 191, pp. 91–96, Sep. 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1871141316301664

[79] B. Mieth, A. Rozier, J. A. Rodriguez, M. M. C. Höhne, N. Görnitz, and K.-R. Müller, "DeepCOMBI: explainable artificial intelligence for the analysis and discovery in genome-wide association studies," *NAR Genomics and Bioinformatics*, vol. 3, no. 3, p. lqab065, Sep. 2021.

[80] G. Montavon, A. Binder, S. Lapuschkin, W. Samek, and K.-R. Müller, "Layer-wise relevance propagation: an overview," *Explainable AI: interpreting, explaining and visualizing deep learning*, pp. 193–209, 2019.

[81] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[82] S. Jaszczur, A. Chowdhery, A. Mohiuddin, Ł. Kaiser, W. Gajewski, H. Michalewski, and J. Kanerva, "Sparse is enough in scaling transformers," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[83] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," 2019.

[84] A. Roy, M. Saffar, A. Vaswani, and D. Grangier, "Efficient Content-Based Sparse Attention with Routing Transformers," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 53–68, Feb. 2021. [Online]. Available: https://doi.org/10.1162/tacl_a_00353

[85] J. Shang, J. Zhang, Y. Sun, and Y. Zhang, "EpiMiner: A three-stage co-information based method for detecting and visualizing epistatic interactions," *Digital Signal Processing*, vol. 24, pp. 1–13, Jan. 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1051200413001838

[86] C. Ponte-Fernández, J. González-Domínguez, and M. J. Martín, "Fiuncho: a program for any-order epistasis detection in cpu clusters," *The Journal of Supercomputing*, pp. 1–20, 2022.

[87] T. Gale, M. Zaharia, C. Young, and E. Elsen, "Sparse gpu kernels for deep learning," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020, pp. 1–14.

[88] M. Bisson and M. Fatica, "A gpu implementation of the sparse deep neural network graph challenge," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, 2019, pp. 1–8.

[89] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 2020, pp. 38–45.

[90] J. W. Rae, A. Potapenko, S. M. Jayakumar, and T. P. Lillicrap, “Compressive transformers for long-range sequence modelling,” *arXiv preprint arXiv:1911.05507*, 2019.

[91] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.

[92] S. Jayakumar, R. Pascanu, J. Rae, S. Osindero, and E. Elsen, “Top-kast: Top-k always sparse training,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 20 744–20 754, 2020.

[93] A. Platzer, “Visualization of snps with t-sne,” *PloS one*, vol. 8, no. 2, p. e56883, 2013.

[94] S. Karamizadeh, S. M. Abdullah, A. A. Manaf, M. Zamani, and A. Hooman, “An overview of principal component analysis,” *Journal of Signal and Information Processing*, vol. 4, 2020.

[95] F. Anowar, S. Sadaoui, and B. Selim, “Conceptual and empirical comparison of dimensionality reduction algorithms (pca, kpca, lda, mds, svd, lle, isomap, le, ica, t-sne),” *Computer Science Review*, vol. 40, p. 100378, 2021.

[96] S. T. Roweis and L. K. Saul, “Nonlinear dimensionality reduction by locally linear embedding,” *science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[97] M. Belkin and P. Niyogi, “Laplacian eigenmaps for dimensionality reduction and data representation,” *Neural computation*, vol. 15, no. 6, pp. 1373–1396, 2003.

[98] V. P. Dwivedi and X. Bresson, “A generalization of transformer networks to graphs,” *arXiv preprint arXiv:2012.09699*, 2020.

[99] M. Phuong and M. Hutter, “Formal algorithms for transformers,” *arXiv*, 2022. [Online]. Available: https://arxiv.org/abs/2207.09238

[100] J. Marchini, P. Donnelly, and L. R. Cardon, “Genome-wide strategies for detecting multiple loci that influence complex diseases,” *Nature genetics*, vol. 37, no. 4, pp. 413–417, 2005.

[101] C.-H. Yang, Y.-D. Lin, L.-Y. Chuang, and H.-W. Chang, “Evaluation of breast cancer susceptibility using improved genetic algorithms to generate genotype snp barcodes,” *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 10, no. 2, pp. 361–371, 2013.

[102] R. Campos, D. Marques, S. Santander-Jiménez, L. Sousa, and A. Ilic, “Heterogeneous cpu+ igpu processing for efficient epistasis detection,” in *European conference on parallel processing*. Springer, 2020, pp. 613–628.

Appendix A

Experimental Results Appendix

In this Appendix, additional results regarding each of the epistasis models analysed in Section 4.4 are presented. Starting with the multiplicative model, Figures A.1 and A.2 present the results for the top 5% and 25% of attention scores for the transformer and its comparison to DeepCOMBI.

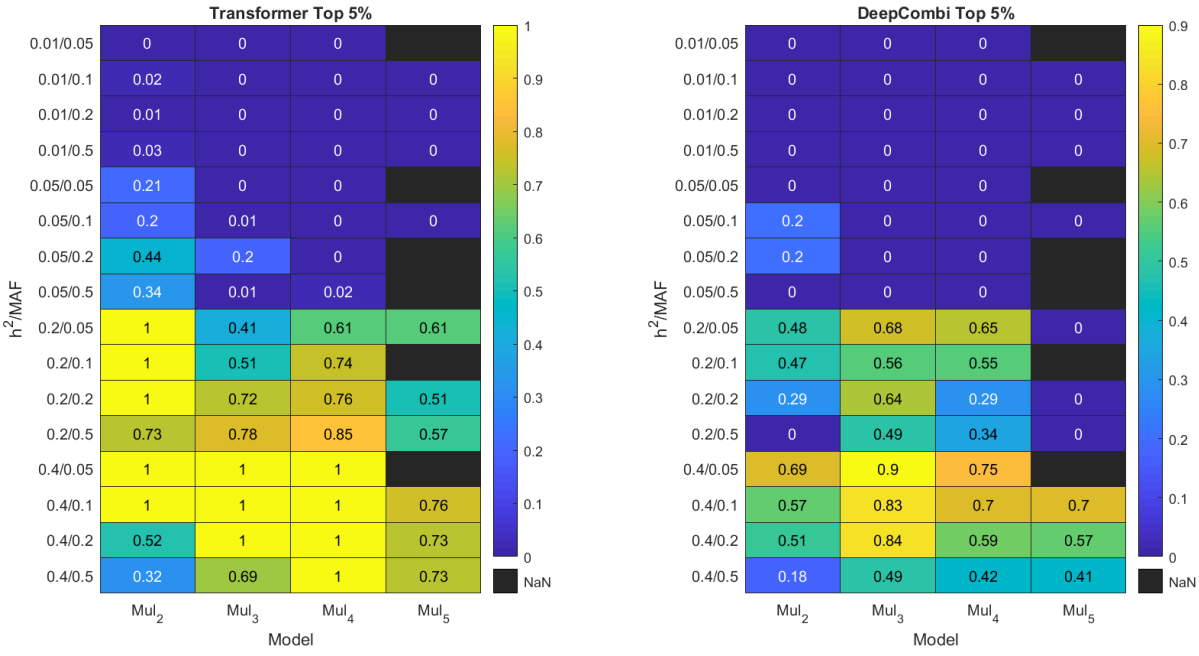

Figure A.1: Multiplicative Model Top 5%

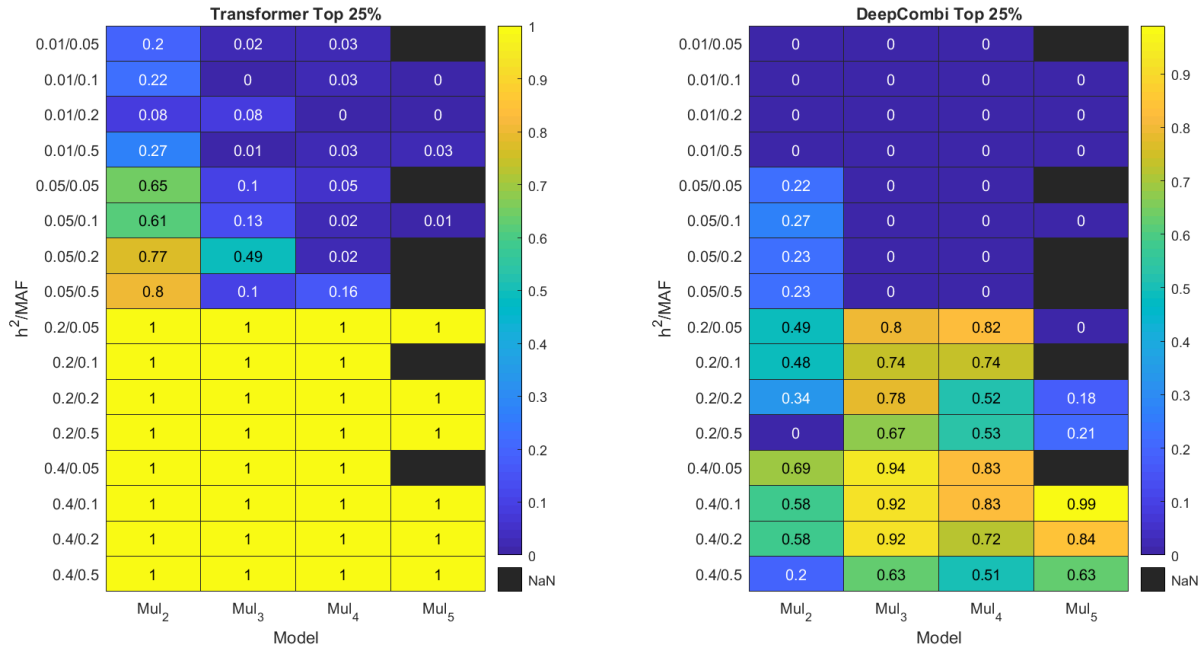

Figure A.2: Multiplicative Model Top 25%

Regarding the performance metrics for the multiplicative model, Figures A.3 to A.5 display the precision, recall, and F1 score, respectively, for the tested networks.

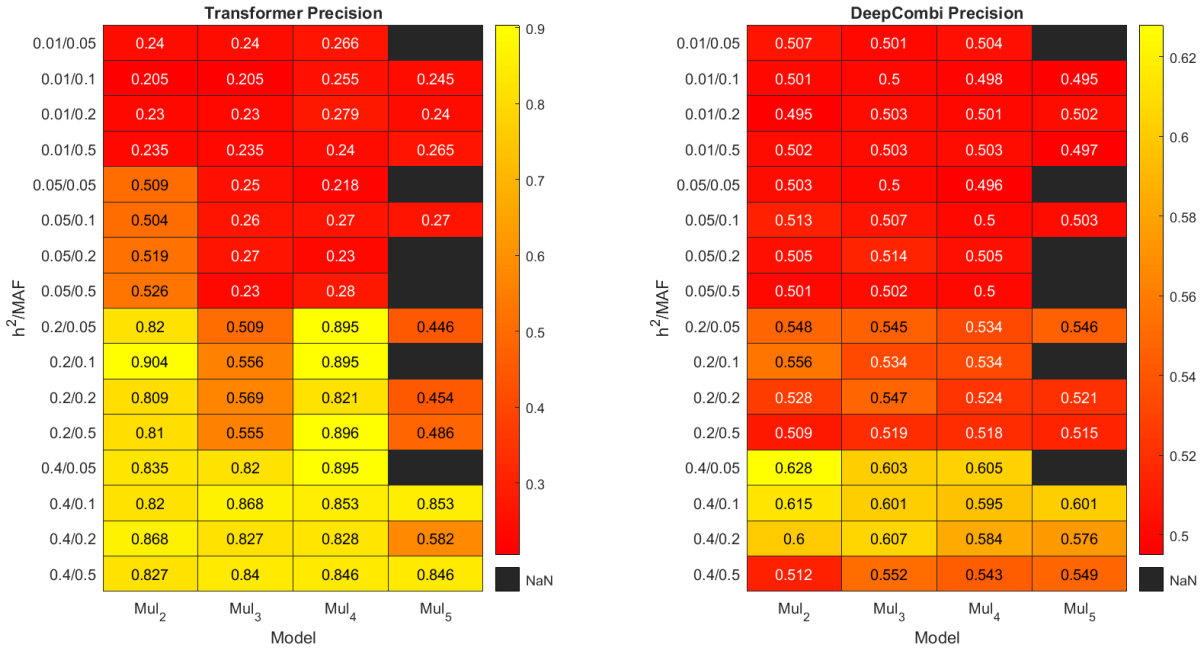

Figure A.3: Multiplicative Model Precision

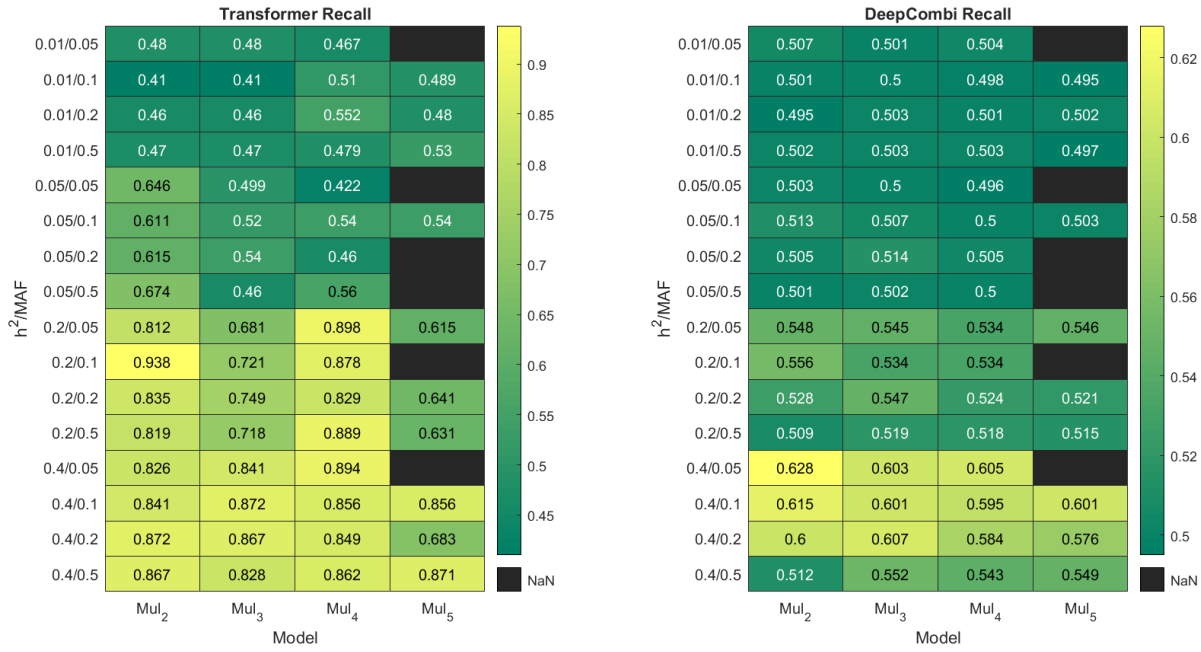

Figure A.4: Multiplicative Model Recall

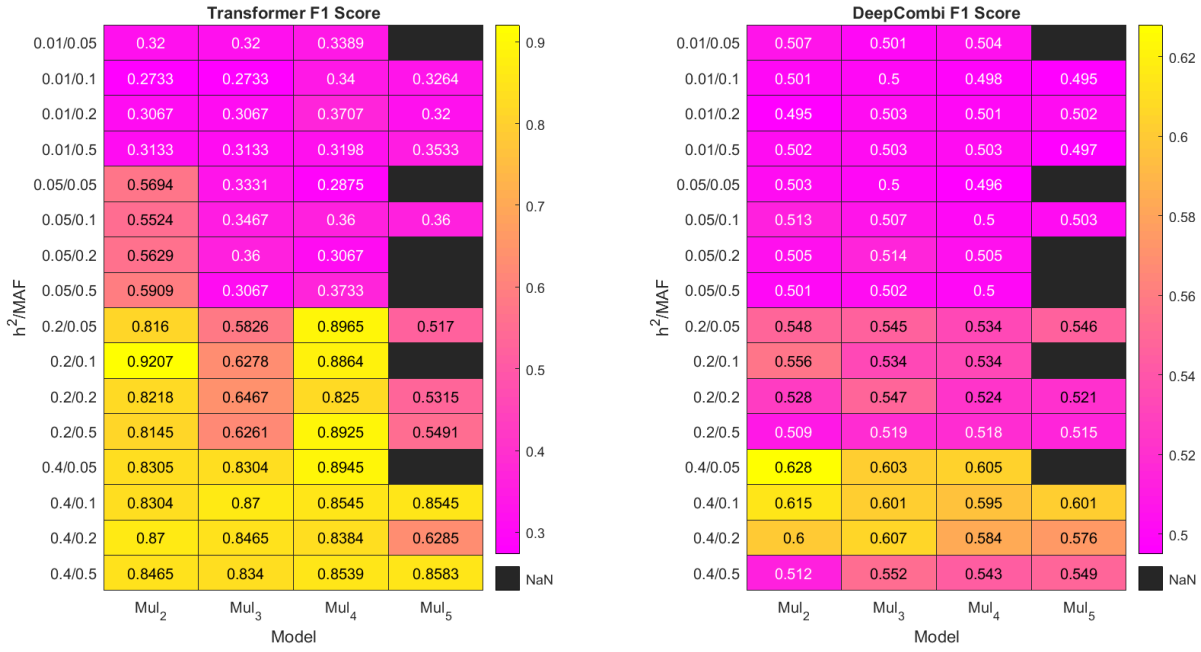

Figure A.5: Multiplicative Model F1 Score

For the threshold model, Figures A.6 and A.7 present the results for the top 5% and 10% of attention scores for the transformer and its comparison to DeepCOMBI.

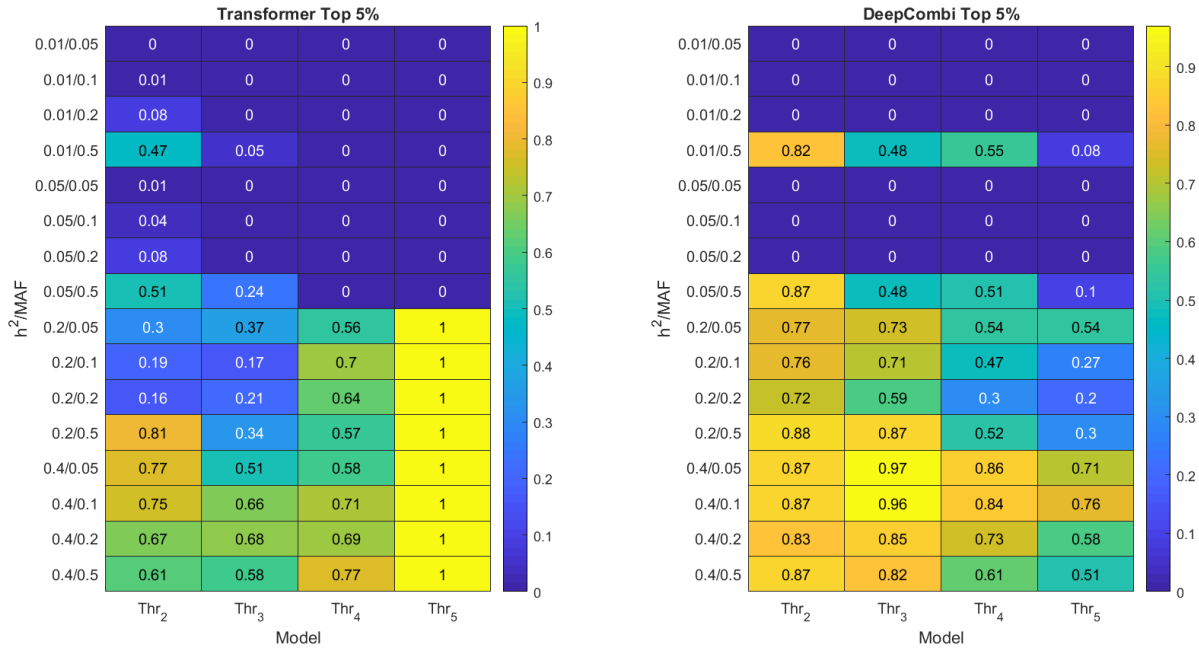

Figure A.6: Threshold Model Top 5%

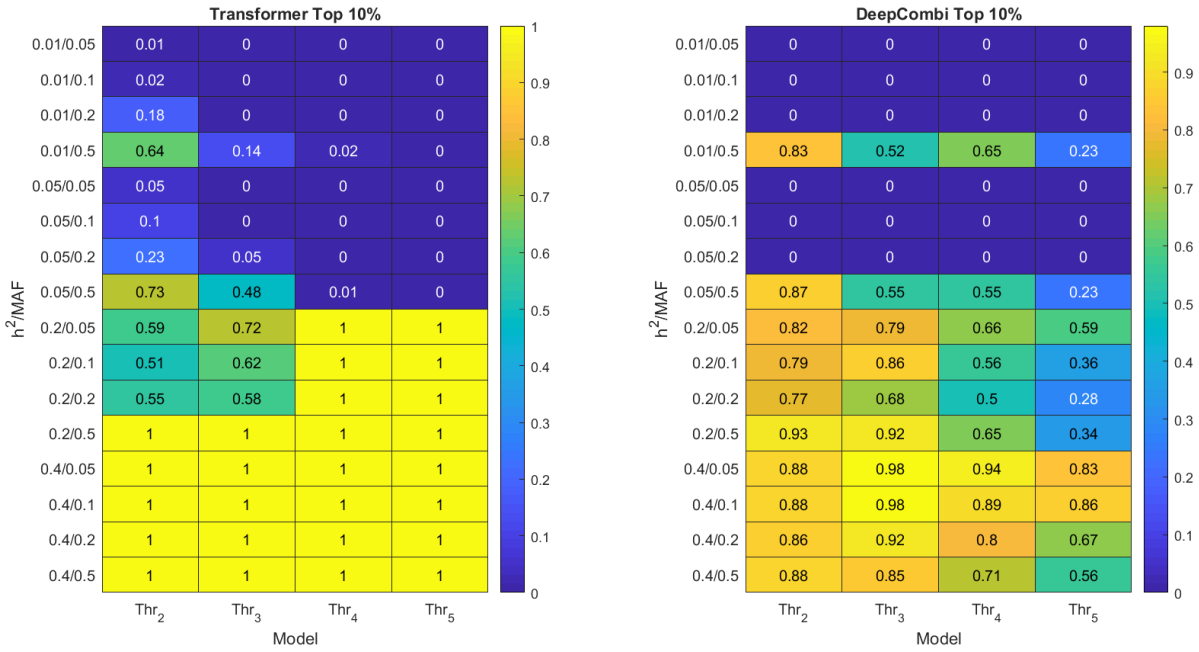

Figure A.7: Threshold Model Top 10%

Regarding the performance metrics for the threshold model, Figures A.8 to A.10 display the precision, recall, and F1 score, respectively, for the tested networks.

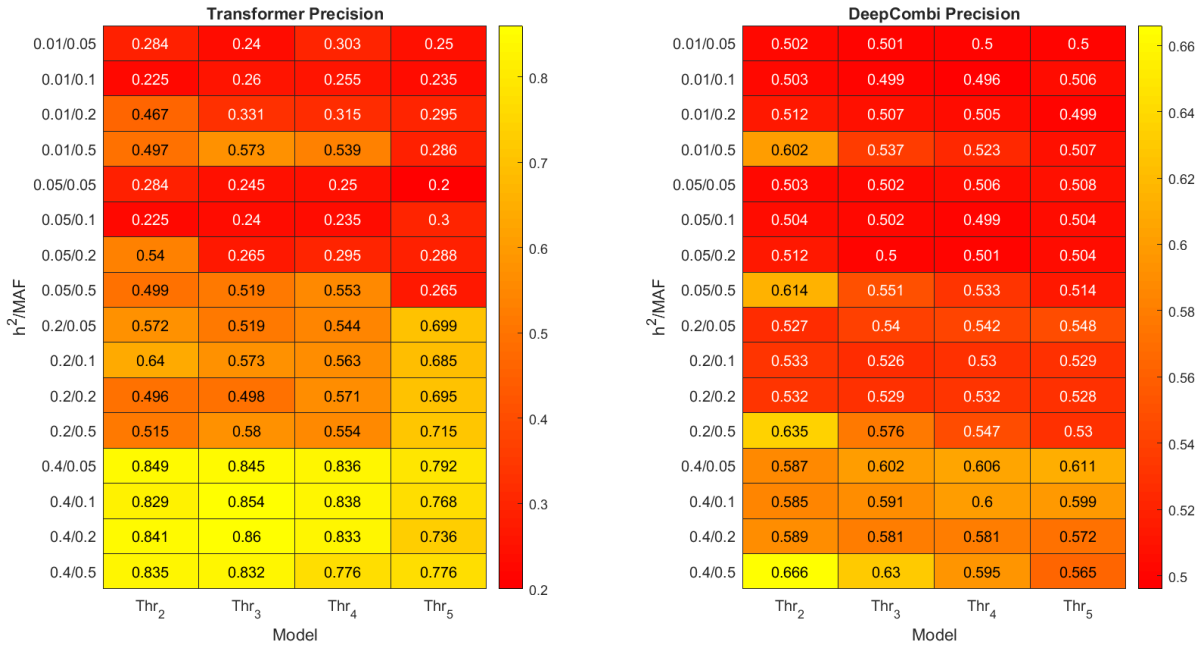

Figure A.8: Threshold Model Precision

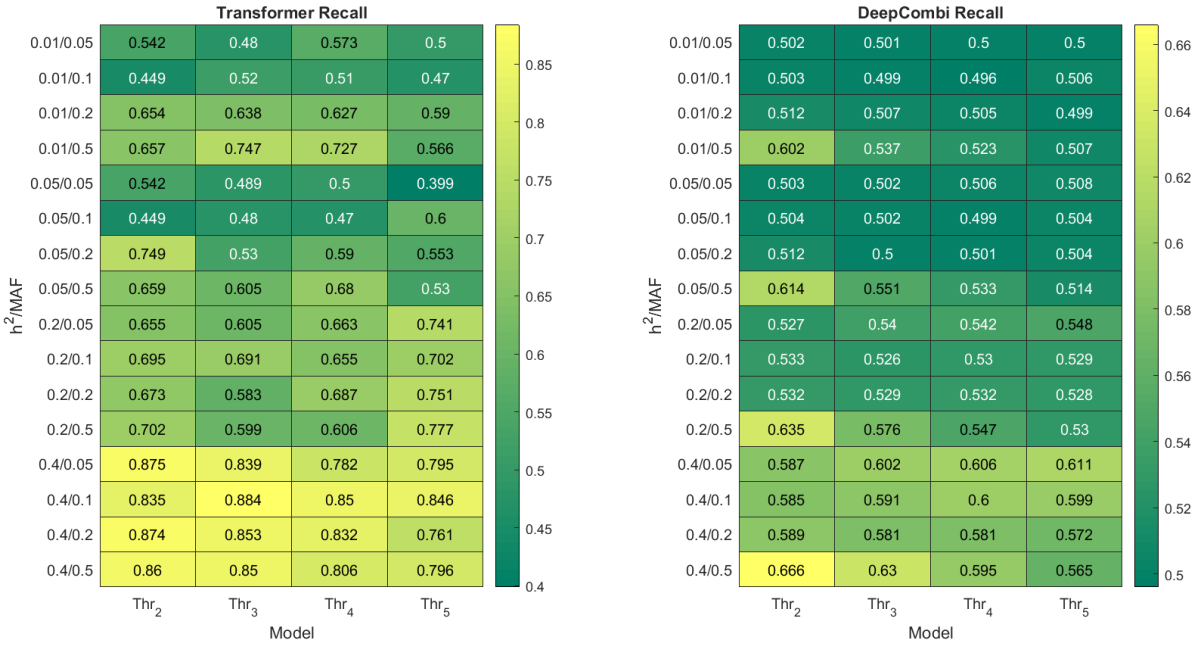

Figure A.9: Threshold Model Recall

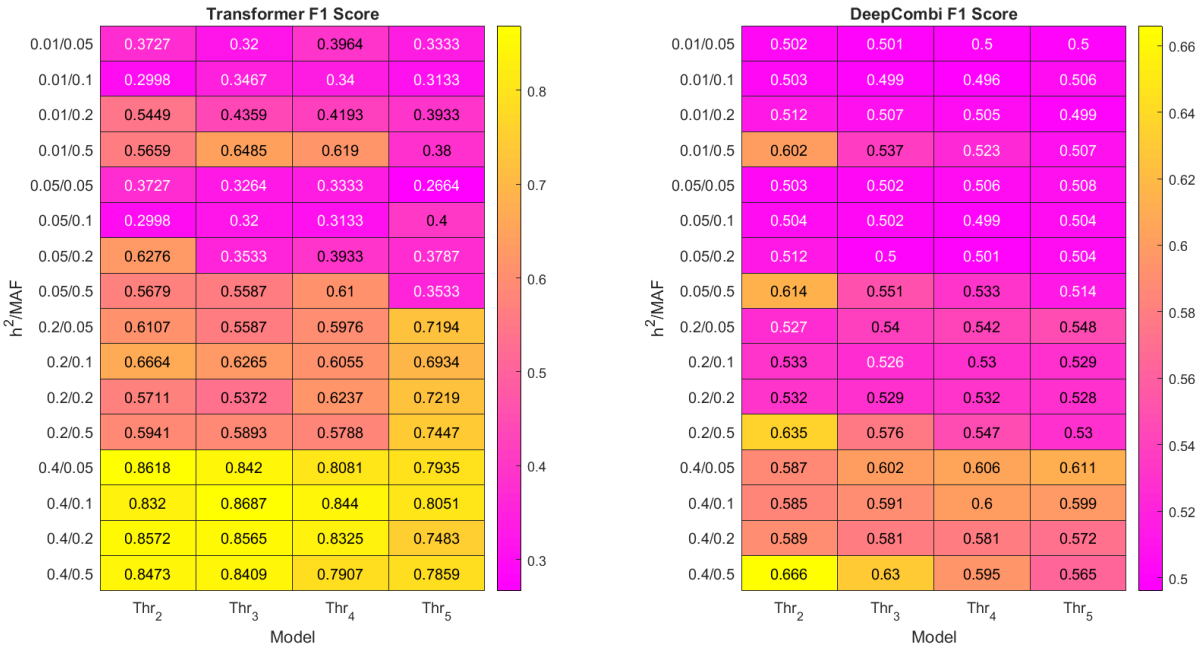

Figure A.10: Threshold Model F1 Score

For the xor model, Figures A.11 and A.12 present the results for the top 5% and 10% of attention scores for the transformer and its comparison to DeepCOMBI.

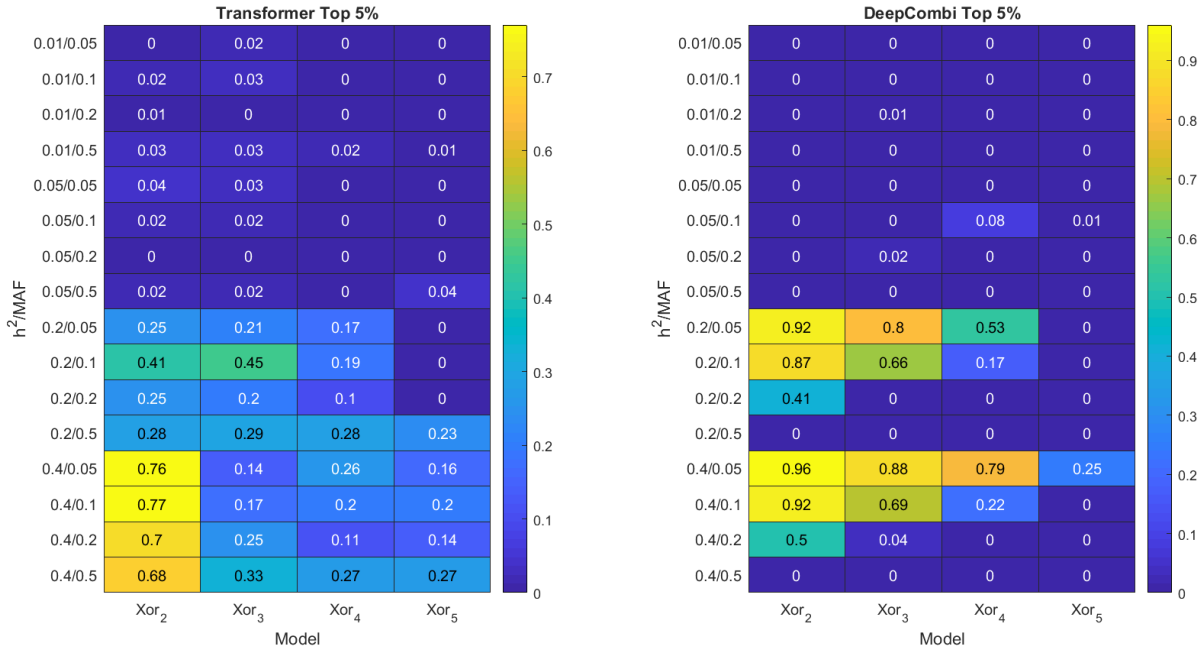

Figure A.11: Xor Model Top 5%

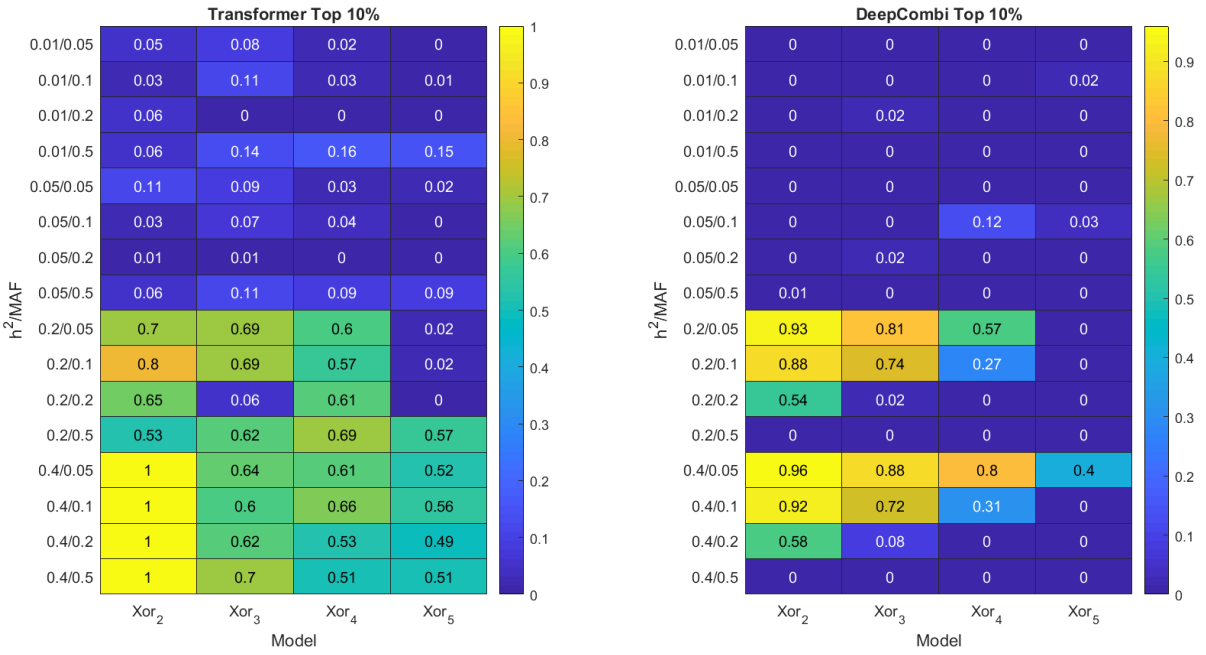

Figure A.12: Xor Model Top 10%

Regarding the performance metrics for the xor model, Figures A.13 to A.15 display the precision, recall, and F1 score, respectively, for the tested networks.

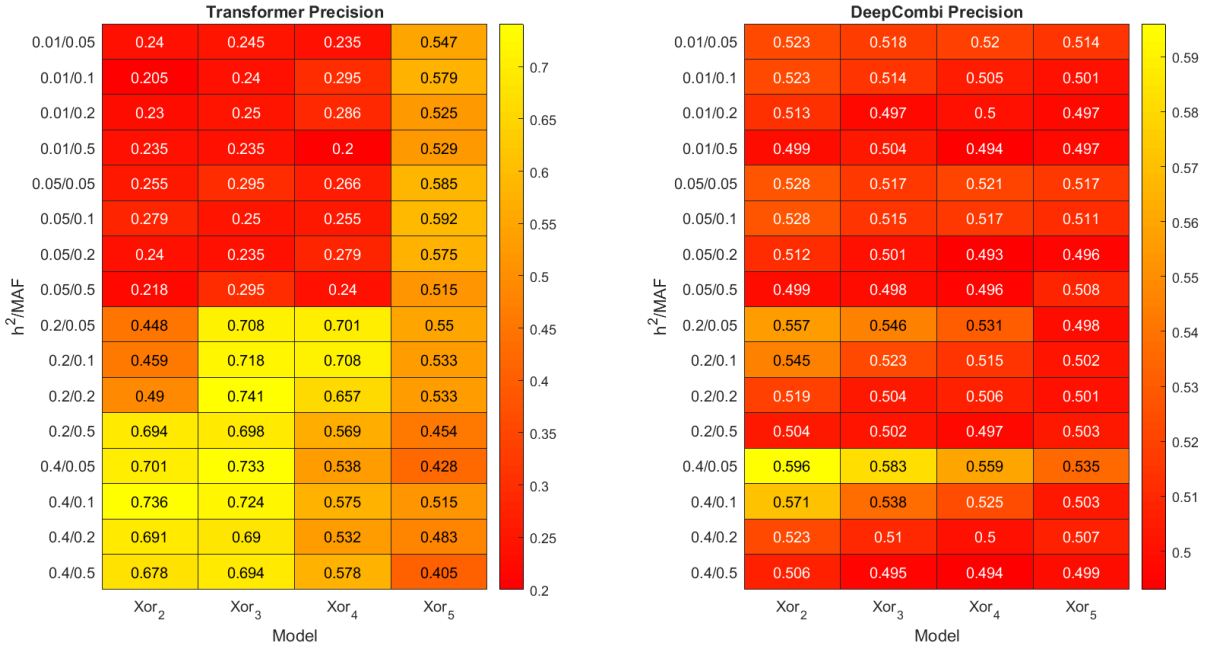

Figure A.13: Xor Model Precision

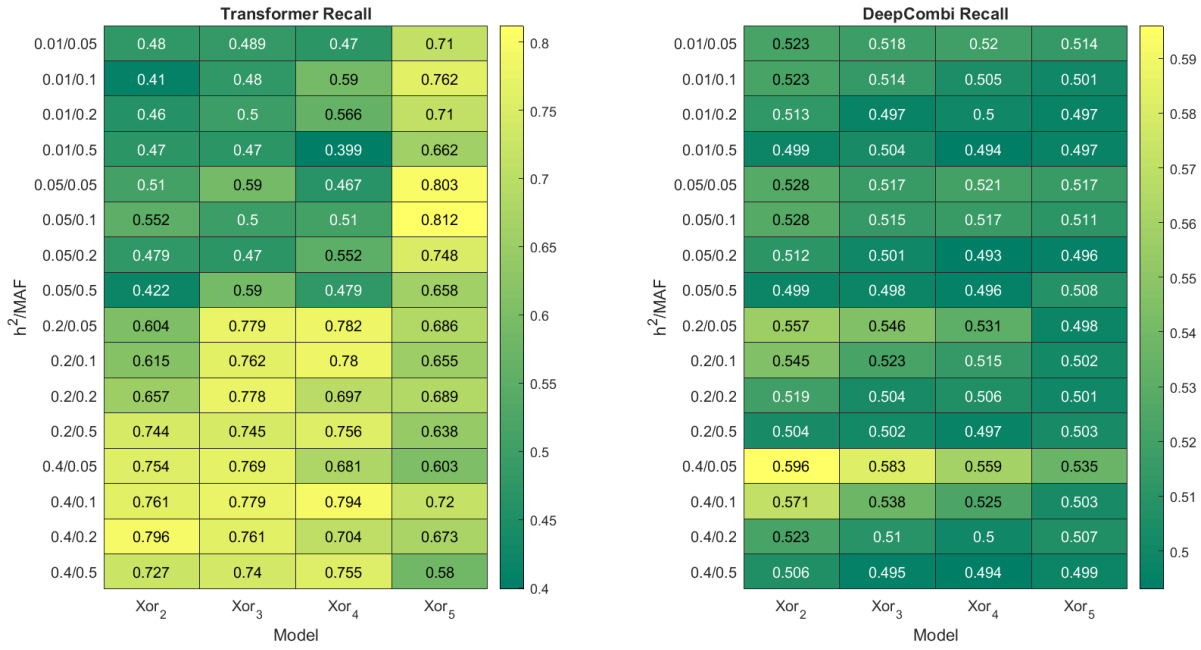

Figure A.14: Xor Model Recall

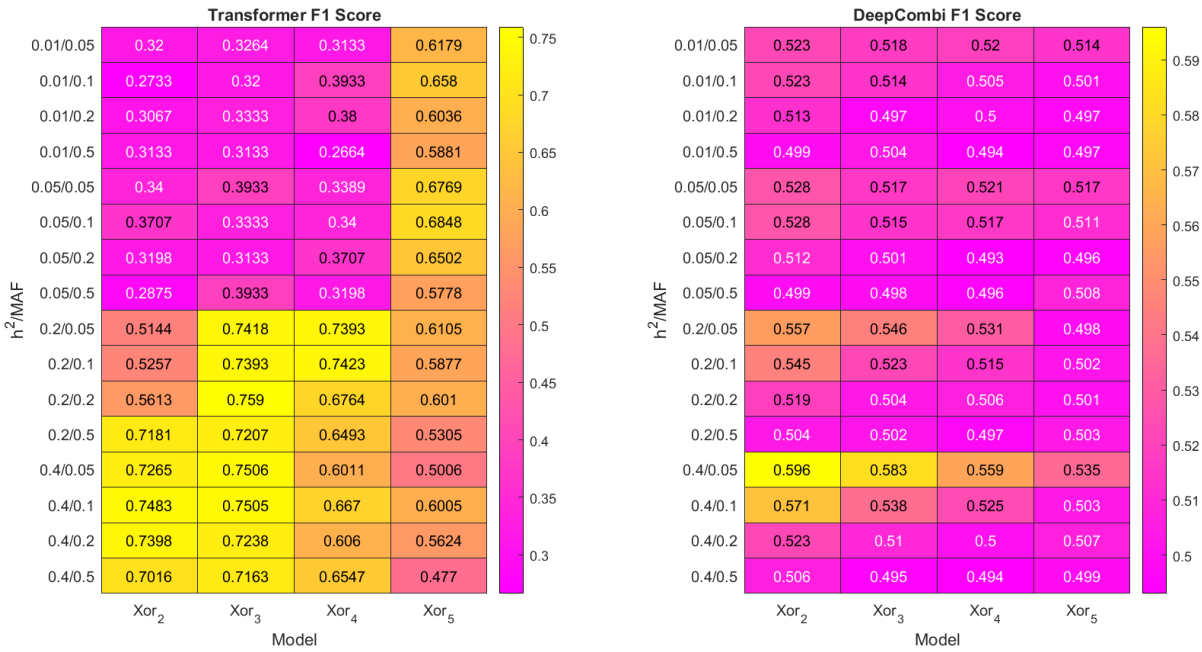

Figure A.15: Xor Model F1 Score