Adapting SNPbag for Phenotype Prediction: A Transformer-Based Approach to Polygenic Risk

Mads Munk Nielsen

Master's thesis, Group MS10-01, Mathematical Statistics

AALBORG UNIVERSITY
STUDENT REPORT

Dept. of Mathematical Sciences
Thomas Manns Vej 23
DK-9220 Aalborg Ø
http://math.aau.dk

Title
Adapting SNPbag for Phenotype Prediction: A Transformer-Based Approach to Polygenic Risk

Theme
Computational Statistics

Project Period
September 2025 - May 2026

Project Group
Group Group MS10-01

Author
Mads Munk Nielsen

Supervisor
Rasmus Waagepetersen

Co-supervisors
Anne Krogh Nøhr
Martin Bøgsted
Rasmus Froberg Brøndum

Page Numbers
47

Date of Completion
May 27, 2026

Abstract

This thesis develops SNPbag, a transformer-based model for binary phenotype prediction from genotype data, and compares it against LDpred2-auto, a Bayesian polygenic risk score method. The model uses a BERT-style encoder trained in two stages: first by reconstructing randomly masked genotypes from their surrounding genomic context, then by fine-tuning for phenotype prediction. Both methods are evaluated on synthetic data from the HAPNEST project, comprising 100000 individuals and 14000 SNPs from chromosome 22, with phenotypes simulated under a logistic disease model with 500 causal variants and 10% prevalence. Pre-training validation accuracy improved from 84.18% at 1000 individuals to 90.19% at 100000, with no signs of overfitting, indicating that the model captures meaningful genomic structure and has not yet saturated. For phenotype prediction, neither method achieved strong discriminative performance: SNPbag obtained a test AUC of 0.54 and LDpred2-auto 0.51, both only marginally above the random baseline of 0.5. These results reflect the severe computational constraints under which both methods operated — SNPbag was fine-tuned for a single epoch with a frozen encoder, and LDpred2-auto was limited to 14000 SNPs from a single chromosome — rather than a fundamental limitation of either approach. Extending the model to longer sequences, more training data, and full encoder fine-tuning are the most important directions for future work.

The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the authors.

Preface

The following report is developed by Mads Munk Nielsen, who studies mathematical statistics in the master's programme at the Department of Mathematical Sciences at Aalborg University. The report was written during the project period September 2025–May 2026.

The code developed in this report is written primarily in *Python* and *R*. The transformer-based model and training pipeline are implemented in *Python* using *PyTorch*, while the polygenic risk score analyses are conducted in *R* using *LDpred2*. Genotype data processing and genome-wide association analyses are performed using *PLINK*. The code is provided as an attachment with the submission and is also available at https://github.com/MadsMunkNielsen/SNPbag-Phenotype-Prediction.

Generative AI was used in this report for language improvement, including improving grammar, clarity, and formulation. All mathematical content, implementation choices, analyses, and conclusions remain the responsibility of the author.

I thank my supervisor Rasmus Waagepetersen and my co-supervisors Anne Krogh Nøhr, Martin Bøgsted, and Rasmus Froberg Brøndum for constructive feedback and guidance during the process.

Study guide

The table of contents of this report is given on page v.

The sources used in the report are referenced on page 45. The references follow the following format:

Author, (Year). Title. Publisher/journal, Edition. Volume(issue):pages

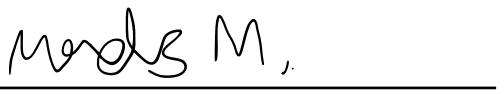
If the edition of the source is not specified, then it is the first edition. If Volume(issue):pages is not specified, then the source is not from a journal. The sources are sorted in alphabetical order by the surname of the author.

Words written as *example* refer to software.

The document was compiled last on May 27, 2026.

Aalborg University, May 27, 2026

Signature



Mads Munk Nielsen

Table of contents

1 Introduction

Understanding why some individuals develop a disease while others do not is one of the central questions in modern medicine. For many common diseases, including diabetes, heart disease, and psychiatric disorders, the answer lies partly in genetics: individuals inherit different combinations of genetic variants, and these variants collectively influence their risk of disease. If genetic risk could be accurately predicted from an individual's DNA, it would enable earlier interventions, targeted screening, and personalised treatment strategies (Torkamani et al., 2018).

Over the past two decades, genome-wide association studies (GWAS) have identified thousands of genetic variants associated with complex traits (Visscher et al., 2017). Each variant typically contributes only a small amount to disease risk, but when aggregated across the genome, these contributions can be substantial. The standard tool for this aggregation is the polygenic risk score (PRS), which sums the estimated effects of many variants weighted by an individual's genotype (Choi et al., 2020). Bayesian approaches such as LDpred2 represent the current state of the art, accounting for the correlation structure between nearby variants and using prior assumptions about the genetic architecture to produce more accurate predictions (Privé et al., 2023).

However, all existing PRS methods share a common limitation: they assume that genetic variants contribute to disease risk in a purely additive manner. There is growing evidence that nonlinear and context-dependent interactions between variants play a role in disease risk (Cordell, 2009), and capturing these interactions requires models that go beyond the additive framework. Transformer models, originally developed for natural language processing (Vaswani et al., 2023), offer a potential path forward. These models learn complex dependencies within sequential data through self-attention, and genetic data has a similar sequential structure: the dependencies between nearby variants, known as linkage disequilibrium, are analogous to the dependencies between words in a sentence (Xu et al., 2025a).

Applying transformers to genetic data is, however, primarily a computational challenge. A typical genome-wide dataset contains millions of variants across hundreds of thousands of individuals, and the memory requirements of transformer models scale with sequence length. Training such models requires significant GPU infrastructure, careful memory management, and practical decisions about batch sizes, sequence lengths, and training schedules that are all constrained by the available hardware. This thesis is therefore a computational statistics project in equal measure to a modelling project: a substantial portion of the work has been dedicated to implementing, profiling, and scaling the training pipeline across multiple GPUs, to data handling and preprocessing at scale, and to understanding the tradeoffs between model capacity, data throughput, and wall-clock time. The modelling decisions described in this thesis are inseparable from these engineering constraints.

The purpose of this thesis is to develop and evaluate SNPbag, a transformer-based model for binary phenotype prediction from genotype data. The evaluation involves comparing the predictive performance of SNPbag against LDpred2-auto (Privé et al., 2023) on simulated phenotypes with known genetic architecture, using classification accuracy and AUC on a shared held-out test set.

2 | Clinical Data and Polygenic Risk Score

This chapter provides the genetic and statistical foundation for genome-wide association studies (GWAS) and polygenic risk scores (PRS). It begins by describing the structure of the human genome and explains how genetic variation is captured through single nucleotide polymorphisms (SNPs). The organisation of SNP-array data is then outlined, together with the quality control steps required before any association analysis can be considered reliable.

As genetic data is correlated and influenced by ancestry, the chapter covers linkage disequilibrium, population structure, and the use of principal component analysis (PCA) to account for these patterns. Lastly, the chapter describes how SNP effects are estimated in GWAS using the methods PLINK and LDpred2 and how these estimated effects are used to compute PRS.

2.1 Genome Structure and Biallelic SNP Encoding

Inside each cell, the nucleus contains most of the deoxyribonucleic acid (DNA), which stores biological information in the specific order of its four chemical bases: adenine (A), guanine (G), cytosine (C), and thymine (T). The bases pair to form complementary matches, and each base on one strand of the DNA molecule aligns with its partner on the opposite strand. Together, the two strands form a spiralling double helix (for Genetic and Services., 2009, pp. 6-9).

In humans, each cell contains 23 pairs of chromosomes, with one chromosome in each pair inherited from each parent. Of these pairs, 22 are autosomes and one is the pair of sex chromosomes. Chromosomes consist of tightly packed DNA and their numbering roughly reflects their size, with lower-numbered chromosomes generally containing more base pairs. Each chromosome pair is homologous, meaning the two chromosomes share the same arrangement of loci. The alleles at these loci, which represent different variants of the same DNA sequence, may differ between the chromosomes, reflecting differences in their underlying nucleotide sequences (Hom, 2023).

Each chromosome carries many genes, which are specific sequences of DNA bases. Across all chromosomes, the total number of genes is approximately 20000 to 25000. The collection of all genes makes up the genome (for Genetic and Services., 2009, pp. 6-9). A phenotype is most often a measurable trait or outcome of an individual, such as height, disease status, or blood pressure. All individuals share approximately 99% of their genetic sequence; the remaining variation contributes to the phenotypic diversity observed between individuals. At specific loci, this variation often appears as SNPs, which are single-base substitutions. Across the roughly three-billion-base human genome, SNPs occur approximately once every 100 to 300 bases (for Genetic and Services., 2009).

A biallelic SNP at locus m has two possible alleles

$$A_m = \{a_{m,1}, a_{m,2}\}, \quad a_{m,1}, a_{m,2} \in \{A, G, C, T\}.$$

Since each individual carries one allele from each parent, there are three possible genotypes

$$\mathcal{G}_m = \{(a_{m,1}, a_{m,1}),\ (a_{m,1}, a_{m,2}),\ (a_{m,2}, a_{m,2})\}.$$

In GWAS, genotypes are commonly encoded numerically, with the genotype of individual i at locus m represented as

$$g_{i,m} \in \{0, 1, 2\},$$

where $g_{i,m} = 0$ indicates homozygosity for the major allele, $g_{i,m} = 1$ indicates heterozygosity, and $g_{i,m} = 2$ indicates homozygosity for the minor allele (Graffelman and Camarena, 2007). The full genotype matrix is $G \in \{0, 1, 2\}^{N \times M}$, with rows indexing individuals and columns indexing SNPs. The m-th column $G_m \in \{0, 1, 2\}^N$ contains the genotypes at SNP m across all N individuals.

2.2 Genotype Data Structure and Quality Control

Genotype data from SNP arrays are typically stored in binary PLINK format (.bed, .bim, .fam files). The .bed file contains the compressed genotype matrix where each row represents an individual and each column represents a SNP. The .bim file provides SNP metadata that includes chromosome, SNP identifier, genetic distance, base-pair position, and the two alleles. The .fam file contains sample information: family ID, individual ID, paternal ID, maternal ID, sex, and phenotype (Marees et al., 2018).

Before an analysis can be conducted, quality control (QC) is needed to ensure reliable results. Some of the QC procedures implemented are:

- **Call rate filtering**: Variants and samples with excessive missing data are removed with a typical threshold of 1% (Marees et al., 2018).
- **Minor allele frequency (MAF)**: SNPs with MAF below a threshold (e.g., 0.05) are excluded to avoid statistical instability from rare variants (Marees et al., 2018).
- **Sex verification**: Reported sex in the .fam file is checked against genetic sex reported in the sex-chromosome (Marees et al., 2018).

Besides the QC, it is also important to consider ancestry and relatedness when conducting genetic analyses. Failure to do so can lead to spurious associations and inflated type I error, as allele frequencies often correlate with both trait distributions and geographic ancestry. Ancestry is accounted for through principal component analysis of the genotype matrix, where the genotypes of individuals with similar ancestry will cluster, see Figure 2.1. These principal components can then be included as covariates in regression models to adjust for population structure, thereby minimizing confounding between genetic and geographic variation (Uffelmann et al., 2021).

2.3 Genome-Wide Association Study and Polygenic Risk Scores

A genome-wide association study (GWAS) estimates the association between each SNP and a phenotype by fitting a separate linear regression for each of the M SNPs. For SNP $m \in \{1, \ldots, M\}$, the model is

$$y = W\alpha + G_m\beta_m + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2 I_N), \tag{2.1}$$

where $y \in \mathbb{R}^N$ is a vector of phenotypes, $W \in \mathbb{R}^{N \times K}$ is a matrix of covariates (including an intercept) with fixed effects $\alpha \in \mathbb{R}^K$, $G_m \in \mathbb{R}^N$ is a vector of genotypes at SNP m, $\beta_m \in \mathbb{R}$

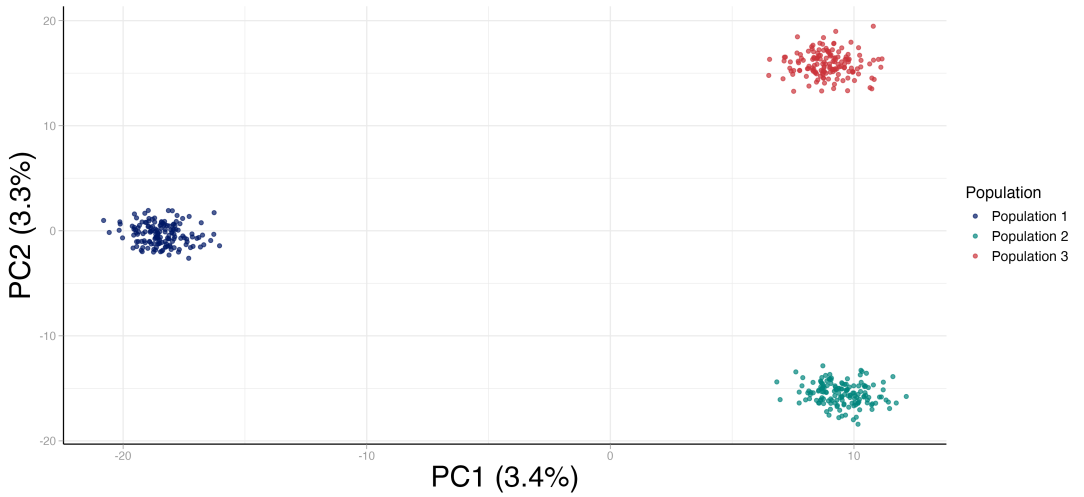


Figure 2.1: Principal component analysis (PCA) of a simulated genotype matrix with $N = 450$ individuals from three ancestral populations and $M = 5000$ SNPs. Each point is one individual, coloured by population. The first two principal components explain 3.4% and 3.3% of the total variance respectively, and clearly separate the three populations into distinct clusters.

is the effect of SNP m on the phenotype, and $\varepsilon \in \mathbb{R}^N$ is a vector of residual errors. In the case of binary phenotypes, the logit link function is used in place of the identity link. This is the model implemented by PLINK2 (Chang et al., 2015, Uffelmann et al., 2021).

The maximum predictive power of any model based on additive genetic effects is bounded by the heritability

$$h^2 = \frac{\sigma_G^2}{\sigma_y^2},$$

where σ_G^2 is the genetic variance and $\sigma_y^2 = \sigma_G^2 + \sigma_\varepsilon^2$ is the total phenotypic variance (Zaitlen and Kraft, 2012).

A Polygenic Risk Score (PRS) aggregates the estimated effects of many SNPs to predict an individual's genetic predisposition to a phenotype. For individual i, the PRS is

$$\mathrm{PRS}_i = \sum_{m \in \mathcal{S}} g_{i,m}\, \hat{\beta}_m,$$

where $\hat{\beta}_m$ is the effect size estimated by the GWAS and $\mathcal{S} \subseteq \{1, \ldots, M\}$ is a subset of SNPs whose effects are statistically significant, typically those with p-value below 5×10^{-8} (Uffelmann et al., 2021).

SNPs that are physically close on the genome tend to be inherited together, which is known as linkage disequilibrium (LD). Their correlation is captured by the LD matrix $R \in \mathbb{R}^{M \times M}$ with elements

$$R_{mk} = \mathrm{Cor}(G_m, G_k).$$

LD is a problem for the simple PRS above: if two correlated SNPs both reflect the same causal signal, including both double-counts that contribution. LDpred2 (Privé et al., 2020) addresses this by re-estimating effect sizes using Bayesian shrinkage that accounts for LD, as described in section 2.4.

2.4 LDpred2: Bayesian Polygenic Prediction

LDpred2 computes PRS as a Bayesian inference problem, deriving posterior effect sizes that account for LD. It depends on summary statistics, $\hat{\beta}_{marg} = (\hat{\beta}_1, \ldots, \hat{\beta}_M)^\top$, and an external LD reference (Privé et al., 2020).

2.4.1 From Marginal Effects to Joint Effects

Fundamental for LDpred2 is the relationship between marginal GWAS effects and joint effects.

Let $S \in \mathbb{R}^{M \times M}$ be a diagonal matrix with standard deviations of M variants, $C_N = I_N - \frac{1}{N}\mathbf{1}\mathbf{1}^T$ be the centering matrix, $G \in \mathbb{R}^{N \times M}$ the genotype matrix, and $y \in \mathbb{R}^N$ a vector of phenotypes. When solving a joint multiple regression model with all variants and an intercept α

$$\begin{bmatrix} \hat{\alpha} \\ \hat{\gamma}_{joint} \end{bmatrix} = \left(\begin{bmatrix} \mathbf{1} & G \end{bmatrix}^T \begin{bmatrix} \mathbf{1} & G \end{bmatrix} \right)^{-1} \begin{bmatrix} \mathbf{1} & G \end{bmatrix}^T y,$$

which simplifies to

$$\hat{\gamma}_{joint} = (G^T C_N G)^{-1} G^T C_N y.$$

The marginal GWAS, by contrast, fits each variant one at a time in a separate regression. The resulting marginal effect estimates on the allele scale are

$$\hat{\gamma}_{marg} = \frac{1}{N-1} S^{-2} G^T C_N y.$$

The correlation matrix of the standardized genotypes is

$$R = \frac{1}{N-1} S^{-1} G^T C_N G S^{-1}.$$

Combining these expressions gives the fundamental relationship between joint and marginal effects

$$\hat{\gamma}_{joint} = S^{-1} R^{-1} S \hat{\gamma}_{marg}. \tag{2.2}$$

In practice, the LD matrix R is either computed directly from the individual-level genotype data when available, or estimated from an external reference panel matched to the ancestry of the GWAS cohort when only summary statistics are available (Privé et al., 2020).

Before computing $\hat{\gamma}_{joint}$ with LDpred2, effect sizes and standard errors must be transformed to a standardized scale. This transformation begins with the estimate $\hat{\gamma}_m$ of the marginal effect for variant m. Let $\tilde{y}$ and $\tilde{G}_m$ denote the phenotype and genotype vectors after residualizing out K covariates, ensuring that $\hat{\gamma}_m$ captures the direct effect of the variant unconfounded by these covariates. The squared standard error of $\hat{\gamma}_m$ is then given by

$$\mathrm{se}(\hat{\gamma}_m)^2 = \frac{(\tilde{y} - \hat{\gamma}_m \tilde{G}_m)^T (\tilde{y} - \hat{\gamma}_m \tilde{G}_m)}{(N - K - 1)\tilde{G}_m^T \tilde{G}_m} \approx \frac{\tilde{y}^T \tilde{y}}{N \tilde{G}_m^T \tilde{G}_m} \approx \frac{\mathrm{var}(y)}{N \,\mathrm{var}(G_m)}, \tag{2.3}$$

where the first approximation holds when the sample size is assumed much larger than the number of covariates ($N \gg K$), and the genetic contribution is weak such that $\tilde{y} \gg \hat{\gamma}_m^2 \tilde{G}_m$.

Solving (2.3) for $\mathrm{sd}(G_m)$ yields

$$\mathrm{sd}(G_m) \approx \frac{\mathrm{sd}(y)}{\mathrm{se}(\hat{\gamma}_m)\sqrt{N}} \implies \underbrace{\mathrm{sd}(G_m) \cdot \hat{\gamma}_m}_{\text{standardized effect}} \approx \frac{\hat{\gamma}_m}{\mathrm{se}(\hat{\gamma}_m)} \frac{\mathrm{sd}(y)}{\sqrt{N}} = \underbrace{\frac{\hat{\gamma}_m}{\mathrm{se}(\hat{\gamma}_m)}}_{Z\text{-score}} \frac{1}{\sqrt{N}}, \tag{2.4}$$

where $\mathrm{sd}(y)$ cancels out in the joint model (2.2) and is therefore set to 1 without loss of generality. With the standard errors from (2.3) and the standardized effect sizes from (2.4), the requirements for LDpred2 have been met (Privé et al., 2020).

2.4.2 Gibbs Sampling

The posterior distribution over the effect sizes $\beta = (\beta_1, \ldots, \beta_M)^\top$ given the observed summary statistics, the LD matrix, and the hyper-parameters does not admit a closed-form solution. LDpred2 approximates the posterior using Gibbs sampling (Gelman et al., 2013, Geman and Geman, 1984).

Gibbs sampling generates samples from a joint distribution $\pi(\beta_1, \ldots, \beta_M)$ by iteratively sampling each component from its full conditional distribution while holding all other components fixed. Concretely, at iteration t, the sampler cycles through $m = 1, \ldots, M$ and draws

$$\beta_m^{(t)} \sim \pi\Big(\beta_m \;\Big|\; \beta_1^{(t)}, \ldots, \beta_{m-1}^{(t)}, \beta_{m+1}^{(t-1)}, \ldots, \beta_M^{(t-1)}\Big),$$

using the most recently sampled values for all other components. Under mild regularity conditions, the sequence of iterates $\{\beta^{(t)}\}_{t\geq 1}$ converges in distribution to the target posterior $\pi(\beta \mid \hat{\beta}_{marg}, R, h^2, \pi_c)$ regardless of the initial values (Gelman et al., 2013).

Because the early iterates are influenced by the initialisation and may not yet reflect the target posterior, the first $N_{\text{burn-in}}$ iterations are discarded. Once the chain has stabilised, the posterior means from the subsequent N_{iter} iterations are accumulated and averaged to produce the final effect size estimates.

2.4.3 The LDpred2 Model

The standardized effect size $\beta_m = S_{m,m}\gamma_m$ relates to the allele-scale effect γ_m through the genotype standard deviation $S_{m,m}$. The prior on β_m is assumed to be

$$\beta_m \sim \begin{cases} \mathcal{N}\left(0, \frac{h^2}{M\pi_c}\right) & \text{with probability } \pi_c, \\ 0 & \text{otherwise.} \end{cases}$$

where $\pi_c \in (0, 1]$ is the proportion of causal variants, M is the total number of variants, h^2 is the SNP heritability, γ_m denotes effects on the allele scale, and $S_{m,m}$ is the standard deviation of the standardized genotype for variant m. This prior uses the assumption that only a fraction π_c of variants contributes to phenotypic prediction, each with variance proportional to the total heritability divided among causal variants. Conditional on the true joint effects β, the sampling distribution of the marginal estimates is approximately normal

$$\hat{\beta}_{marg} \mid \beta \sim \mathcal{N}(R\beta, V),$$

where R is the LD matrix and $V = \mathrm{diag}\left(\frac{1}{N}\right)$ approximates the variance of the marginal effect estimates for large sample sizes N, assuming standardized genotypes. The posterior

distribution $p(\beta \mid \hat{\beta}_{marg}, R, h^2, \pi_c)$ lacks a closed-form solution due to the spike-and-slab structure of the prior; LDpred2 therefore uses the Gibbs sampler described in subsection 2.4.2, iteratively drawing each β_m from its conditional posterior given all other effects β_{-m} and the observed data (Privé et al., 2020).

The conditional sampling for variant m depends on the residualized marginal effect

$$\tilde{\beta}_m = \hat{\beta}_m - \sum_{k \neq m} R_{mk}\beta_k = \hat{\beta}_m - R_{m,-m}\beta_{-m},$$

where $R_{m,-m}$ denotes the m-th row of R excluding the diagonal element $R_{mm} = 1$, and β_{-m} represents the vector of current effect estimates with β_m excluded. This removes the effects that variant m borrows from correlated neighbours through LD, leaving only the evidence attributable to variant m itself (Privé et al., 2020).

Given the residualized effect $\tilde{\beta}_m$, the posterior probability that variant m is causal is obtained by applying Bayes' theorem to the spike-and-slab prior. The two hypotheses are compared: under the causal hypothesis ($\gamma_m = 1$), $\tilde{\beta}_m$ was drawn from the normal component $\mathcal{N}(0, h^2/M\pi_c)$; under the non-causal hypothesis ($\gamma_m = 0$), $\tilde{\beta}_m$ should be zero. The resulting posterior probability is

$$\bar{p}_m = \frac{1}{1 + \frac{1-\pi_c}{\pi_c}\sqrt{1 + \frac{Nh^2}{M\pi_c}}\exp\left(-\frac{1}{2} \cdot \frac{N\tilde{\beta}_m^2}{1+\frac{M\pi_c}{Nh^2}}\right)}.$$

When the residualized effect is large, the exponential term vanishes and $\bar{p}_m$ approaches 1; when it is close to zero, $\bar{p}_m$ falls back toward the prior π_c.

The effect size β_m is then sampled from a two-component mixture: with probability $\bar{p}_m$ it is drawn from a normal distribution centred near $\tilde{\beta}_m$ but shrunk toward zero, and with probability $1 - \bar{p}_m$ it is set to exactly zero

$$\beta_m \mid \tilde{\beta}_m \sim \begin{cases} \mathcal{N}\left(\frac{\tilde{\beta}_m}{1+\frac{M\pi_c}{Nh^2}}, \frac{1}{N\left(1+\frac{M\pi_c}{Nh^2}\right)}\right) & \text{with probability } \bar{p}_m, \\ 0 & \text{with probability } 1 - \bar{p}_m. \end{cases}$$

Averaging over both cases gives the posterior mean

$$\mathbb{E}[\beta_m \mid \tilde{\beta}_m] = \bar{p}_m \cdot \frac{\tilde{\beta}_m}{1 + \frac{M\pi_c}{Nh^2}},$$

which serves as the point estimate for prediction. The Gibbs sampler iterates through all M variants, updating each effect conditional on the current values of all others. After discarding the first $N_{\text{burn-in}}$ iterations, the posterior means from the remaining N_{iter} iterations are averaged to produce the final effect estimates. Algorithm 2.1 summarises the complete procedure (Privé et al., 2020).

The output of 2.1 is a vector of posterior mean effect sizes $\hat{\beta}^{\text{LDpred2}} \in \mathbb{R}^M$ on the allele scale. These replace the naive marginal GWAS estimates in the PRS formula: for individual i, the LDpred2 polygenic risk score is

$$\text{PRS}_i^{\text{LDpred2}} = \sum_{m=1}^{M} g_{i,m}\,\hat{\beta}_m^{\text{LDpred2}}.$$

Note that the summation runs over all M variants, not a thresholded subset $\mathcal{S}$, because the Bayesian shrinkage has already driven non-causal effects toward zero.

Algorithm 2.1 LDpred2 Gibbs Sampler with hyper-parameters π_c and h^2

$\hat{\beta} \leftarrow (\hat{\gamma}/\text{se}(\hat{\gamma})) \cdot \sqrt{N}$ ▷ *Initialize standardized marginal effects*
$\Omega \leftarrow 0$ ▷ *Initialize posterior means accumulator*
for $t = 1, \ldots, N_{\text{burn-in}} + N_{\text{iter}}$ **do**
  for each variant m **do**
    Compute $\tilde{\beta}_m = \hat{\beta}_m - \sum_{k \neq m} R_{mk}\beta_k$ ▷ *Residualize against other variants*
    Compute $\bar{p}_m$ ▷ *Posterior probability of causality*
    Sample $\gamma_m \sim \text{Bernoulli}(\bar{p}_m)$
    if $\gamma_m = 1$ **then**
      Sample $\beta_m \sim \mathcal{N}\left(\frac{\tilde{\beta}_m}{1+\frac{M\pi_c}{Nh^2}}, \frac{1}{N\left(1+\frac{M\pi_c}{Nh^2}\right)}\right)$
    else
      $\beta_m \leftarrow 0$
    $\omega_m \leftarrow \bar{p}_m \cdot \frac{\tilde{\beta}_m}{1+\frac{M\pi_c}{Nh^2}}$ ▷ *Posterior mean contribution*
  if $t > N_{\text{burn-in}}$ **then**
    $\Omega \leftarrow \Omega + \omega$ ▷ *Accumulate after burn-in*
$\Omega \leftarrow \Omega / N_{\text{iter}}$ ▷ *Average posterior means*
return $\Omega \cdot \text{se}(\hat{\gamma})/\sqrt{N}$ ▷ *Scale back to original units*

2.4.4 LDpred2-auto: Automatic Hyper-parameter Estimation

The LDpred2 model requires specification of two hyper-parameters: the proportion of causal variants π_c and the SNP heritability h^2. LDpred2-auto extends the model by estimating both directly within the Gibbs sampling procedure, eliminating the need for a grid search or an external validation dataset (Privé et al., 2020).

The heritability is estimated from the current state of the sampler as

$$\hat{h}^2 = \hat{\beta}^T R \hat{\beta},$$

where $\hat{\beta}$ denotes the vector of current effect sizes. For the causal proportion, let $M_c = \sum_m \mathbb{1}(\beta_m \neq 0)$ denote the number of non-zero variants in the current iteration. With a uniform prior $\pi_c \sim \text{Beta}(1, 1)$, conjugacy gives

$$\pi_c \mid M_c \sim \text{Beta}(1 + M_c, 1 + M - M_c),$$

from which π_c is sampled directly at each iteration. Both estimates are updated after each complete pass through all variants, allowing the model to adaptively learn the genetic architecture from the data itself (Privé et al., 2020).

To assess convergence, LDpred2-auto is run as multiple independent chains with diverse initialisations of π_c on a logarithmic scale. Chains that diverge are discarded, and the effect sizes from the remaining chains are averaged or selected based on validation performance to produce the final estimates (Privé et al., 2020).

3 Architecture and Training of SNPbag for Phenotype Prediction

This chapter describes the architecture of SNPbag (Xu et al., 2025a) and explains how it is adapted for phenotype prediction. The chapter is structured to follow the data through the model: from raw genotypes to embeddings, through the encoder, and into a multi-layer perceptron (MLP) used for either predicting masked genotypes or phenotypes. The architecture of the adapted model is seen in Figure 3.1.

Throughout this chapter, the model contains learnable parameters such as weight matrices and bias vectors. The reader should not be concerned with how these parameters are obtained at this stage, they are treated as given. How the parameters are learned through training is described in section 3.5.

The first step in the model is to map genotypes and loci into continuous vector representations known as embeddings. For each position in the sequence, the genotype and the corresponding locus are each mapped to a separate learned vector in $\mathbb{R}^{512}$, and the two vectors are combined to form a single input representation. Each such position is referred to as a token, that is a single discrete input element in the sequence. This embedding procedure is described in section 3.1.

The embedded sequence serves as input for the Bidirectional encoder representations from transformers (BERT)-based transformer encoder described in section 3.2. The encoder consists of $L = 16$ stacked transformer layers, each applying multi-head self-attention followed by a feed-forward network. Through these layers, each token's representation is progressively refined to incorporate information from the entire genomic context, capturing both linear and nonlinear associations between SNPs.

The output from the BERT encoder is used as input to two different MLPs: one used for masked genotype prediction during pre-training, and one used for phenotype prediction during fine-tuning. During pre-training, the model is trained using masked genotype prediction, where genotype tokens are randomly masked with a fixed probability and the model learns to reconstruct the original genotypes from their surrounding genomic context. To achieve this, an MLP maps the encoder output at each position to three logits, one for each genotype, $\{0, 1, 2\}$. The model parameters are then optimized by minimizing a cross-entropy loss computed from these logits and the corresponding true genotype. By training the model end-to-end, this prediction task updates both the genotype prediction MLP and the encoder, encouraging the encoder to learn representations that capture dependencies between SNPs.

Following pre-training, the model is adapted for phenotype prediction. The pre-trained encoder is reused, but the masked genotype prediction MLP is replaced by a separate phenotype prediction MLP. Since the phenotype is defined for an entire individual rather than for a single SNP position, the encoder outputs are first aggregated across the sequence dimension to form one representation per individual. This individual-level representation is then passed to the phenotype MLP, which maps it to a single logit used for binary phenotype prediction. Both prediction MLPs are described in section 3.3.

The training procedure is described in section 3.5. This includes the construction of training, validation, and test splits, the use of batches and epochs, the computation of gradients by backpropagation, and the parameter updates performed by the AdamW optimizer. Finally, section 3.6 describes the metrics used to assess masked genotype prediction during pre-training and phenotype prediction during fine-tuning.

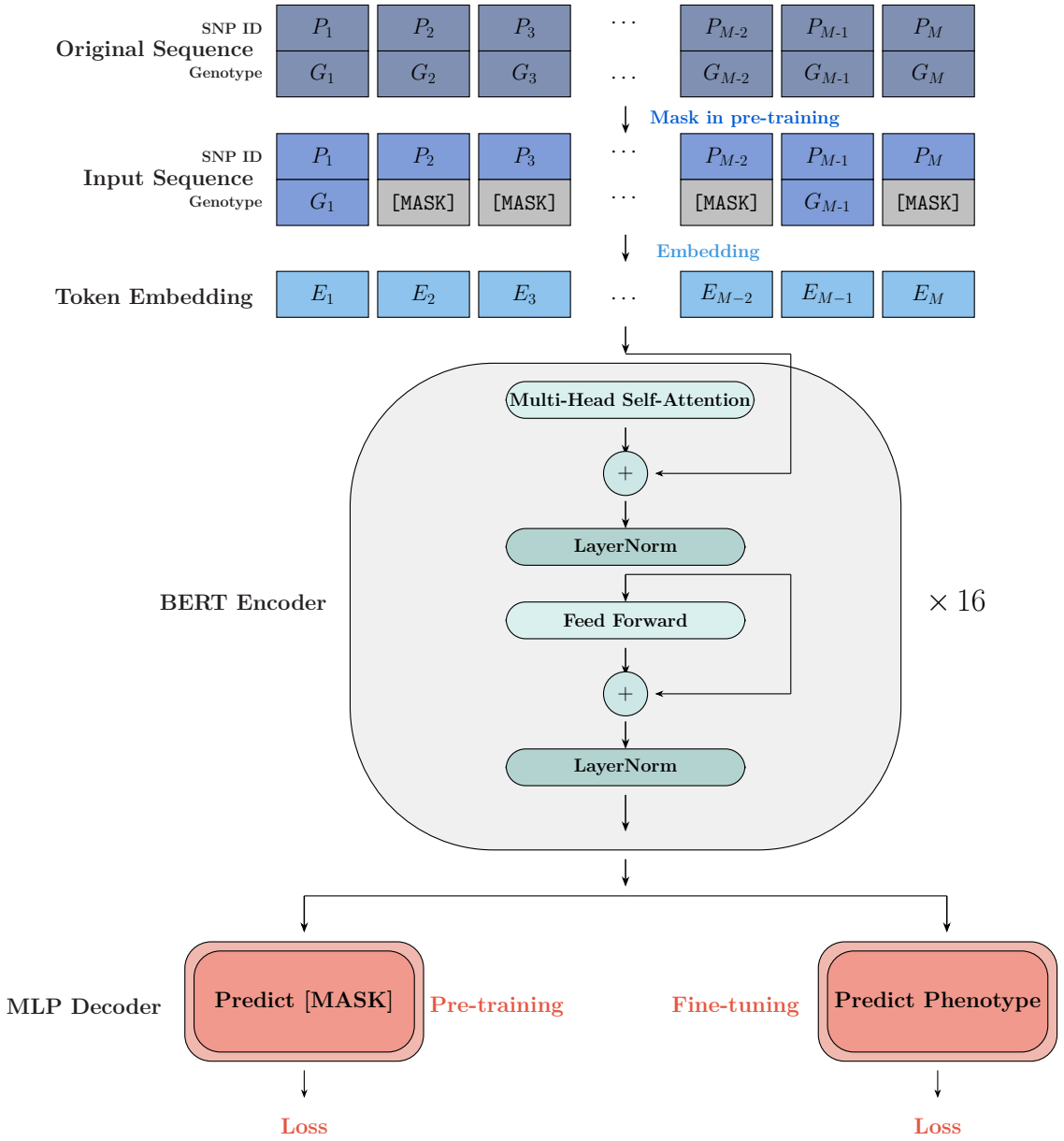


Figure 3.1: Architecture of the SNPbag model. The original genotype sequence is masked during pre-training, as described in subsection 3.3.1, and embedded, as described in section 3.1. The embedded sequence is used as input to a BERT encoder, described in section 3.2, consisting of a stack of $L = 16$ transformer encoder layers. Each encoder layer contains multi-head self-attention, described in subsection 3.2.1, residual connections with layer normalization, described in subsection 3.2.3, and a feed-forward network, described in subsection 3.2.2. The encoder output is passed to two MLPs prediction heads, described in section 3.3: one for masked genotype prediction during pre-training, described in subsection 3.3.1, and one for phenotype prediction during fine-tuning, described in subsection 3.3.2.

3.1 Construction of Input Embeddings

Consider $N \in \mathbb{N}$ individuals, each with $M \in \mathbb{N}$ observed genotypes. Let $G \in \{0,1,2\}^{N\times M}$ denote the genotype matrix, and let $P \in \mathbb{N}^M$ denote the vector of loci, such that the entry $G_{i,m}$ represents the genotype of individual i at SNP m, located at locus P_m. While this representation is natural, transformer models require their inputs as continuous vectors in $\mathbb{R}^d$, thus the genotypes and loci are first mapped into an embedding space. By mapping genotypes to an embedding space, the model learns a representation in which relationships between genotypes are determined by the data. To achieve this, each genotype $G_{i,m}$ is first represented as a one-hot encoded vector

$$G_{i,m} \mapsto e(G_{i,m}),$$

where $e(0) = (1,0,0)$, $e(1) = (0,1,0)$, and $e(2) = (0,0,1)$. For individual i, the M one-hot vectors are stacked row-wise into a matrix $e(G_i) \in \mathbb{R}^{M\times 3}$, and the genotype embedding is obtained by applying a learnable linear transformation to this collection

$$\eta_i = e(G_i)\, W_G, \quad \eta_i \in \mathbb{R}^{M\times d}$$

where $W_G \in \mathbb{R}^{3\times d}$ is a trainable weight matrix and d denotes the embedding dimension, which henceforth is fixed to $d = 512$. The resulting embeddings across all N individuals are collected into a tensor

$$E_G = \begin{pmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_N \end{pmatrix} \in \mathbb{R}^{N\times M\times 512},$$

where each $\eta_i \in \mathbb{R}^{M\times 512}$ (Boué, 2025).

The same approach is used to obtain positional embeddings from the loci vector. Since the M loci are shared across all individuals, each position $j \in \{1,\dots,M\}$ is mapped to a learnable embedding via

$$P_m = e_m^\top W_P,$$

where $e_m \in \mathbb{R}^M$ is the m-th standard basis vector and $W_P \in \mathbb{R}^{M\times 512}$ is a trainable weight matrix. Equivalently, p_m is simply the m-th row of W_P. The M positional embeddings are stacked row-wise into a matrix $\tilde{E}_P \in \mathbb{R}^{M\times 512}$, which is then replicated across all N individuals to form a tensor $E_P \in \mathbb{R}^{N\times M\times 512}$.

With both genotype and positional embeddings defined, the genotype embeddings are first scaled by $\sqrt{512}$ to ensure their magnitudes are comparable to the positional embeddings before addition (Vaswani et al., 2023). The final representation is then constructed as

$$E = \sqrt{512}\, E_G + E_P. \tag{3.1}$$

The resulting embedding tensor $E \in \mathbb{R}^{N\times M\times 512}$ is used as input to the BERT encoder, see Figure 3.1(Xu et al., 2025b).

Example 3.1. Construction of input embedding
Consider $N = 2$ individuals and $M = 4$ SNPs with genotype matrix and loci vector

$$G = \begin{pmatrix} 0 & 1 & 2 & 1 \\ 1 & 0 & 1 & 2 \end{pmatrix}, \quad P = (101, 305, 502, 718).$$

For individual 1, the genotypes $(0, 1, 2, 1)$ are one-hot encoded and stacked row-wise into

$$e(G_1) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \in \mathbb{R}^{4\times 3}.$$

Multiplying by the weight matrix $W_G \in \mathbb{R}^{3\times 512}$ yields the genotype embedding $\eta_1 = e(G_1)\, W_G \in \mathbb{R}^{4\times 512}$, where each row is the learned representation of the corresponding genotype. In this case, since the rows of $e(G_1)$ are standard basis vectors, the m-th row of η_1 simply selects the row of W_G corresponding to genotype $G_{1,m}$. Similarly, individual 2 has genotypes $(1, 0, 1, 2)$, giving

$$e(G_2) = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \in \mathbb{R}^{4\times 3},$$

and $\eta_2 = e(G_2)\, W_G \in \mathbb{R}^{4\times 512}$. The genotype embeddings are then stacked into $E_G = (\eta_1, \eta_2)^\top \in \mathbb{R}^{2\times 4\times 512}$. For the positional embeddings, each SNP position $m \in \{1, 2, 3, 4\}$ selects the m-th row of $W_P \in \mathbb{R}^{4\times 512}$, yielding $\tilde{E}_P = W_P \in \mathbb{R}^{4\times 512}$. This matrix is replicated for both individuals to obtain $E_P \in \mathbb{R}^{2\times 4\times 512}$. Finally, the input embedding is constructed as $E = \sqrt{512}\, E_G + E_P \in \mathbb{R}^{2\times 4\times 512}$. □

3.2 Bidirectional Encoder Representations from Transformers

The BERT encoder processes each individual independently. For individual $i \in \{1, \ldots, N\}$, the input is the embedding matrix $E_i \in \mathbb{R}^{M\times 512}$ defined in (3.1), where each row is the 512-dimensional embedding of a single SNP. Each row of E_i is the 512-dimensional embedding of a single SNP. At this stage, each row depends only on the genotype and position of that SNP, it contains no information about the surrounding genomic context. The role of the encoder is to transform these initial, context-free embeddings into context-aware representations, where each row incorporates information from the other SNPs in the sequence.

The encoder is built as a stack of $L = 16$ transformer layers, each with the same architecture but its own learnable parameters. For each individual, the input embedding E_i enters the first layer, and the representation produced by one layer is passed as input to the next, see Figure 3.1. To track the representation as it flows through the encoder, the following notation is used. Let $a^{(\ell)} \in \mathbb{R}^{M\times 512}$ denote the output of encoder layer ℓ, with $a^{(0)} \coloneqq E_i$ being the input to the first layer and $a^{(L)}$ being the final encoder output. Within each layer ℓ, the transformation from $a^{(\ell-1)}$ to $a^{(\ell)}$ proceeds through four intermediate steps, each producing a matrix in $\mathbb{R}^{M\times 512}$:

1. **Multi-head self-attention** (subsection 3.2.1) produces $s^{(\ell)}$. This is where tokens interact: each SNP gathers information from other SNPs based on learned attention weights. It is the only operation that creates dependencies between positions.

2. **Residual connection and layer normalisation** (subsection 3.2.3) produces $z^{(\ell)}$. The attention output $s^{(\ell)}$ is added to the input $a^{(\ell-1)}$ and the result is normalised. The addition ensures that the original representation is preserved alongside the new information.

3. **Feed-forward network** (subsection 3.2.2) produces $f^{(\ell)}$. A nonlinear transformation is applied independently to each token, allowing the model to learn nonlinear patterns within each SNP's representation.

4. **Second residual connection and layer normalisation** produces $a^{(\ell)}$. The feed-forward output $f^{(\ell)}$ is added to $z^{(\ell)}$ and normalised, yielding the final output of layer ℓ.

In summary, the data flows through each layer ℓ as

$$a^{(\ell-1)} \xrightarrow{\text{attention}} s^{(\ell)} \xrightarrow{\text{residual + norm}} z^{(\ell)} \xrightarrow{\text{feed-forward}} f^{(\ell)} \xrightarrow{\text{residual + norm}} a^{(\ell)}.$$

After $L = 16$ layers, the final output $a^{(L)} \in \mathbb{R}^{M\times 512}$ is a matrix where each row is a context-aware representation of the corresponding SNP, shaped by the information from all other SNPs in the sequence. The following subsections describe each operation in detail. The following subsections describe each operation in detail. The focus is on the architecture, what each operation computes and how the representations are transformed. How the learnable parameters in these operations are determined through training is described separately in section 3.5.

3.2.1 Multi-Head Self-Attention

Self-attention is a mechanism that allows each element in a sequence to attend to every other element, calculating attention weights that represent dependencies between each token in the sequence. It is the only operation in the transformer encoder that creates interactions between SNPs. The multi-head extension runs several self-attention mechanisms, each with its own learnable parameters, allowing the mechanism to capture different types of relationships between SNPs. The mechanism is first described for a single attention head and then extended to the multi-head setting.

At each of the $L = 16$ encoder layers, the input $a^{(\ell-1)} \in \mathbb{R}^{M\times 512}$ is transformed to queries, keys, and values

$$Q = a^{(\ell-1)}\, W_Q^{(\ell)}, \quad K = a^{(\ell-1)}\, W_K^{(\ell)}, \quad V = a^{(\ell-1)}\, W_V^{(\ell)},$$

where $W_Q^{(\ell)}, W_K^{(\ell)} \in \mathbb{R}^{512\times d_k}$ and $W_V^{(\ell)} \in \mathbb{R}^{512\times d_v}$ are learnable weight matrices, with each encoder layer maintaining its own independent set of parameters. Although Q, K, and V are all computed from the same input $a^{(\ell-1)}$, they differ because the three weight matrices $W_Q^{(\ell)}$, $W_K^{(\ell)}$, and $W_V^{(\ell)}$ are initialised with different random values and are updated independently during training, causing each to learn a different transformation of the input. The output of the head is then

$$\text{Attention}(Q, K, V) = \text{softmax}\!\left(\frac{Q\,K^\top}{\sqrt{d_k}}\right) V \;\in\; \mathbb{R}^{M\times d_v},$$

where $QK^\top \in \mathbb{R}^{M\times M}$ computes the pairwise attention scores between all tokens. The softmax function, $\text{softmax}(x)_j = \exp(x_j)/\sum_k \exp(x_k)$, is applied to each row independently so that the entries sum to one and can be interpreted as weights. Because the exponential amplifies differences between its inputs, large entries in $QK^\top$ dominate the softmax output, causing the resulting weights to concentrate on a few tokens. As the dimension d_k grows, the empirical variance of the dot products in $QK^\top$ grows proportionally. Large inputs to the softmax cause it to concentrate nearly all its weight on a single token. Dividing by

$\sqrt{d_k}$ normalises the variance back to a stable scale, allowing the softmax to distribute its weight across multiple positions rather than collapsing onto one (Vaswani et al., 2023). Multiplying by V produces a weighted average of the value vectors for each token, where tokens with higher attention weights contribute more to the output.

A single attention head can only capture one type of relationship between tokens. To capture multiple types of relationships simultaneously, $n_h = 16$ heads are run in parallel within each encoder layer. Each head h applies the single-head attention mechanism described above with its own linear transformation matrices $W_{Q_h}^{(\ell)} \in \mathbb{R}^{512\times d_k}$, $W_{K_h}^{(\ell)} \in \mathbb{R}^{512\times d_k}$, $W_{V_h}^{(\ell)} \in \mathbb{R}^{512\times d_v}$. Since each head's weight matrices are initialised with different random values, they follow different trajectories during training and converge to different solutions, allowing each head to specialise in a different type of dependency between tokens. The n_h head outputs are then concatenated and linearly transformed

$$s^{(\ell)} = \text{MultiHead}(a^{(\ell-1)}) = \text{concat}(\text{head}_1, \ \ldots, \ \text{head}_{n_h})\, W_O^{(\ell)} \ \in \ \mathbb{R}^{M\times 512},$$

where each $\text{head}_h = \text{Attention}(a^{(\ell-1)}W_{Q_h}^{(\ell)},\ a^{(\ell-1)}W_{K_h}^{(\ell)},\ a^{(\ell-1)}W_{V_h}^{(\ell)})$ and $\text{concat}(\cdot)$ stacks the n_h head outputs column-wise into a single matrix in $\mathbb{R}^{M\times(n_h\cdot d_v)}$. Since each of the $n_h = 16$ heads produces an output in $\mathbb{R}^{M\times d_v}$ with $d_k = d_v = 512/n_h = 32$, the concatenation lies in $\mathbb{R}^{M\times(n_h\cdot d_v)} = \mathbb{R}^{M\times 512}$. The output matrix $W_O^{(\ell)} \in \mathbb{R}^{512\times 512}$ then maps this concatenation back to $\mathbb{R}^{M\times 512}$, mixing information across heads so that each token's representation can draw on the different relationship types that the individual heads have captured(Vaswani et al., 2023).

Example 3.2. Multi-head self-attention
Consider a single individual with a sequence of $M = 4$ SNPs with $n_h = 2$ attention heads, each with $d_k = d_v = 2$. Suppose that, after transformation, the first head has query, key, and value matrices

$$Q_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix}, \quad K_1 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, \quad V_1 = \begin{pmatrix} 2 & 1 \\ 0 & 3 \\ 1 & 1 \\ 4 & 0 \end{pmatrix}.$$

The raw score matrix is

$$Q_1 K_1^\top = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

After scaling by $\sqrt{d_k} = \sqrt{2}$ and applying the row-wise softmax, the attention weights become

$$\text{softmax}\left(\frac{Q_1 K_1^\top}{\sqrt{2}}\right) \approx \begin{pmatrix} 0.33 & 0.33 & 0.17 & 0.17 \\ 0.17 & 0.33 & 0.33 & 0.17 \\ 0.19 & 0.35 & 0.27 & 0.19 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{pmatrix}.$$

Each row now sums to one over the four SNPs. For instance, the third row shows that SNP 3 assigns the highest weight (0.35) to SNP 2, showing a large compatibility score between

them. The head output is obtained by multiplying these weights with the value matrix:

$$\text{head}_1 = \text{softmax}\Big(\frac{Q_1 K_1^\top}{\sqrt{2}}\Big) V_1 \approx \begin{pmatrix} 1.34 & 1.49 \\ 0.84 & 1.83 \\ 1.08 & 1.62 \\ 1.75 & 1.25 \end{pmatrix}.$$

Each row of the output is a weighted average of the value vectors, where the weights are determined by the learned compatibility between tokens. SNP 3, for example, draws most heavily from the value vector of SNP 2, so its output representation is influenced primarily by the information that SNP 2 carries.

Now suppose head 2, using its own parameters, independently learned because they are initialized with different random values and updated independently during training, yields

$$\text{head}_2 \approx \begin{pmatrix} 0.50 & 2.10 \\ 1.20 & 0.80 \\ 0.90 & 1.40 \\ 1.00 & 1.00 \end{pmatrix}.$$

The two heads are concatenated column-wise and form the matrix

$$\text{concat}\big[\text{head}_1,\ \text{head}_2\big] \approx \begin{pmatrix} 1.34 & 1.49 & 0.50 & 2.10 \\ 0.84 & 1.83 & 1.20 & 0.80 \\ 1.08 & 1.62 & 0.90 & 1.40 \\ 1.75 & 1.25 & 1.00 & 1.00 \end{pmatrix} \in \mathbb{R}^{4\times 4}.$$

Finally, the output matrix $W_O \in \mathbb{R}^{4\times 4}$ is applied. Without this transformation, the columns of each head's output would remain non-interacting. Multiplying by W_O produces linear combinations across all four columns, yielding $\text{MultiHead}(a^{(0)}) \in \mathbb{R}^{4\times 4}$, where each row now blends the representations learned by both heads. □

3.2.2 Feed-Forward Network

The multi-head self-attention mechanism described in subsection 3.2.1 allows each token to gather information from the entire sequence, but the operation is solely linear. The feed-forward network complements this by applying a nonlinear transformation independently to each token, allowing the model to also learn nonlinear interactions.

The feed-forward network has the same architecture in all 16 encoder layer, but each layer has its own independently learned weight matrices and bias vectors. Within a given layer, the same weights and biases are applied to each of the M token outputs independently. For a single token representation $z_m^{(\ell)} \in \mathbb{R}^{512}$, the transformation is

$$f_m^{(\ell)} = \text{FFN}(z_m^{(\ell)}) = W_2\, \sigma(W_1\, z_m^{(\ell)} + b_1) + b_2 \ \in\ \mathbb{R}^{512},$$

where $W_1 \in \mathbb{R}^{d_{\text{ff}}\times 512}$ and $W_2 \in \mathbb{R}^{512\times d_{\text{ff}}}$ are learnable weight matrices, $b_1 \in \mathbb{R}^{d_{\text{ff}}}$ and $b_2 \in \mathbb{R}^{512}$ are bias vectors, d_{ff} is the dimension of the hidden layer (commonly set to $d_{\text{ff}} = 2048$), and $\sigma(\cdot)$ is a nonlinear activation function applied element-wise. Each encoder layer has its own independent set of feed-forward parameters (Vaswani et al., 2023).

The activation function used in BERT is often chosen as the Gaussian Error Linear Unit (GELU), given as

$$\text{GELU}(x) = x \cdot \Phi(x), \tag{3.2}$$

where $\Phi(x)$ is the cumulative distribution function of the standard normal distribution. Thus, for each element, x, GELU multiplies x by $\Phi(x)$, which is the probability that a standard normal random variable falls below x. For large positive inputs $\Phi(x)$ approaches one, so the output is approximately x itself. For large negative inputs $\Phi(x)$ approaches zero, so the output is approximately 0. The function is depicted in Figure 3.2.

Since $\Phi(x)$ has no closed-form expression, the following approximation is used in practice

$$\mathrm{GELU}(x) \approx 0.5\,x\left(1+\tanh\left[\sqrt{\frac{2}{\pi}}\left(x+0.044715\,x^3\right)\right]\right),$$

where $\tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$ is the hyperbolic tangent (Hendrycks and Gimpel, 2016).

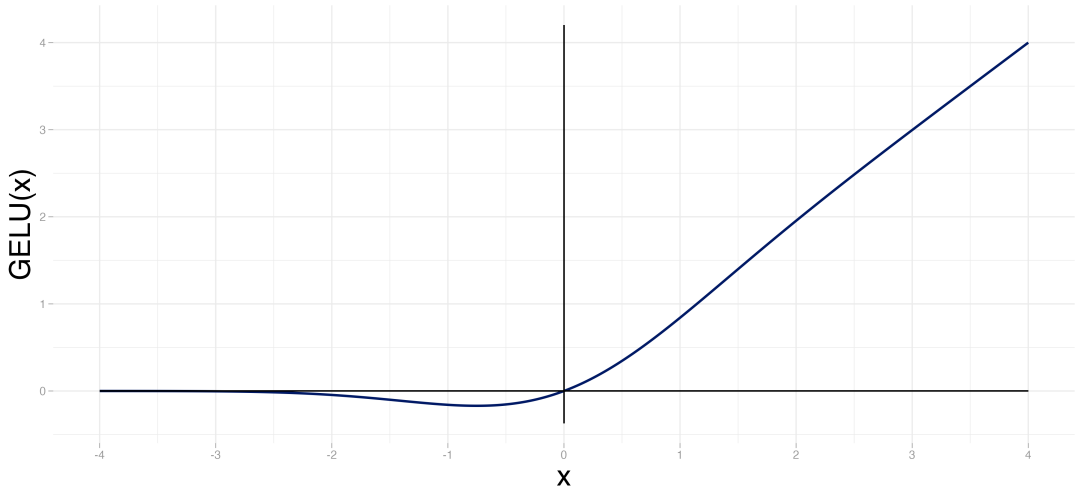


Figure 3.2: The GELU activation function. For large positive x the function approaches the identity $y = x$, while for large negative x the output approaches zero.

The feed-forward network does not introduce any interaction between tokens each row is processed individually. This is intentional as the preceding multi-head self-attention has already mixed information across the sequence, so each token's representation already reflects the full genomic context. The feed-forward network then transforms these representations individually, adding nonlinearity to features that attention has produced (Vaswani et al., 2023).

3.2.3 Residual Connections and Layer Normalization

As the complexity of the encoder increases, the input that each layer receives has already been transformed multiple times, making it increasingly difficult for gradients to propagate back to the earlier layers during training. Residual connections address this by carrying the original input directly to the output of each sublayer in the encoder (He et al., 2015).

Each of the $L = 16$ encoder layers contains two sublayers: the multi-head self-attention mechanism described in subsection 3.2.1, and the feed-forward network described in subsection 3.2.2. A residual connection wraps each sublayer, followed by layer normalization

$$z^{(\ell)} = \mathrm{LayerNorm}\big(a^{(\ell-1)} + s^{(\ell)}\big),$$
$$a^{(\ell)} = \mathrm{LayerNorm}\big(z^{(\ell)} + f^{(\ell)}\big),$$

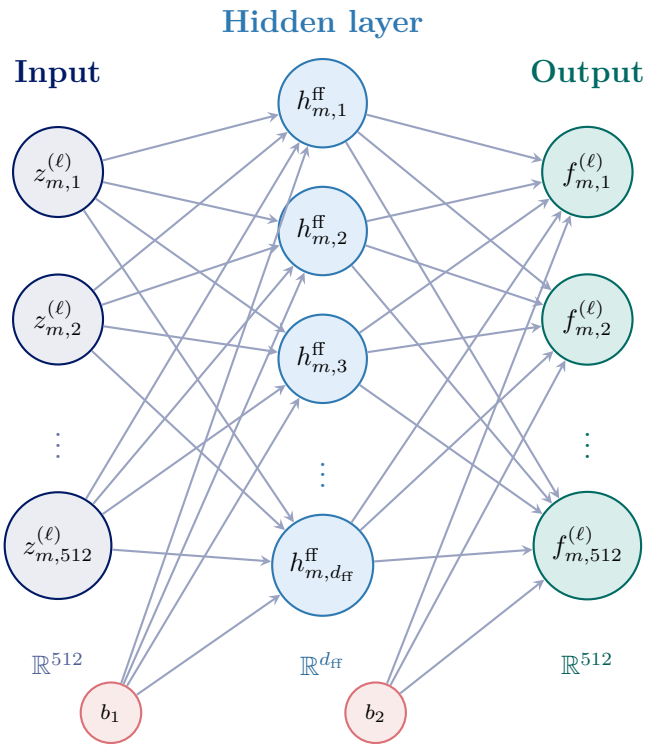


Figure 3.3: Visualization of the feed-forward network for a single token .$z_m^{(\ell)}$ at encoder layer ℓ. The input $z_m^{(\ell)} \in \mathbb{R}^{512}$ is mapped into $\mathbb{R}^{d_{\text{ff}}}$ by W_1 and the activation function $h_m^{\text{ff}} = \sigma(W_1 z_m^{(\ell)} + b_1)$. The output $f_m^{(\ell)} = W_2 h_m^{\text{ff}} + b_2 \in \mathbb{R}^{512}$ is then obtained by mapping back to $\mathbb{R}^{512}$ via W_2. The same parameters are shared across all M tokens.

where $s^{(\ell)} = \text{MultiHead}(a^{(\ell-1)})$ is the output of the multi-head self-attention sublayer, $f^{(\ell)} = \text{FFN}(z^{(\ell)})$ is the output of the feed-forward sublayer, $z^{(\ell)}$ is the output of the first residual connection and normalization, and $a^{(\ell)} \in \mathbb{R}^{M \times 512}$ is the final output of layer ℓ.

Layer normalization is applied after each residual connection, as the repeated addition of sublayer outputs would otherwise grow with each layer in the encoder. For each token m, the operation normalizes across the 512 embedding dimensions

$$\text{LayerNorm}(u_m) = \gamma \odot \frac{u_m - \mu_m}{\sqrt{\sigma_m^2 + \varepsilon}} + \beta, \tag{3.3}$$

where u_m denotes the m-th row of the summed output from either residual addition, i.e. $a^{(\ell-1)} + s^{(\ell)}$ or $z^{(\ell)} + f^{(\ell)}$, μ_m and σ_m^2 are the empirical mean and variance computed across its 512 components, ε is a small constant included for numerical stability (often set to 10^{-9}), $\odot$ denotes element-wise multiplication, and $\gamma, \beta \in \mathbb{R}^{512}$ are learnable parameters that rescale and shift each dimension after normalization, initialized to $\gamma = \mathbf{1}$ and $\beta = \mathbf{0}$ (Ba et al., 2016).

3.3 Prediction Multi-Layer Perceptrons

The BERT encoder described in section 3.2 produces, for each individual i, a matrix of contextualized representations $a^{(L)} \in \mathbb{R}^{M \times 512}$, where each row $a_m^{(L)} \in \mathbb{R}^{512}$ encodes information about SNP m informed by the entire sequence. These representations are learned through self-supervised pre-training: a subset of genotype positions is masked

before embedding, and a genotype prediction MLP maps the encoder output at each masked position to three genotype logits. The model is then trained end-to-end by minimizing the cross-entropy loss between these logits and the true genotypes, using no phenotype labels. This forces the encoder to learn representations that capture dependencies between SNPs.

After pre-training, the genotype prediction MLP is discarded and replaced by a separate phenotype prediction MLP. The pre-trained encoder weights are retained, so the new MLP operates on representations that already encode genomic structure. Both MLPs are depicted in Figure 3.4 (Xu et al., 2025a).

3.3.1 Masked Genotype Prediction

During pre-training (Devlin et al., 2019), each genotype is replaced by a mask token with an 85% probability before the input embedding is constructed. The model's task is to recover the original genotype values from the surrounding context, forcing the encoder to learn representations that capture dependencies between SNPs without requiring any phenotypic labels.

The recovery is framed as a classification problem: at each masked position m, the model estimates the probability that the original genotype belongs to each of the three classes $\{0, 1, 2\}$. These probabilities are modelled as a softmax transformation of three real-valued scores, called logits, which are in turn produced by an MLP. Concretely, for each masked position m, the corresponding encoder output row $a_m^{(L)} \in \mathbb{R}^{512}$ is passed through a linear transformation into $\mathbb{R}^{2048}$, followed by the GELU activation, and a second linear transformation to $\mathbb{R}^3$, yielding three genotype logits

$$h_m^{\text{geno}} = \sigma(a_m^{(L)} W_1 + b_1) \ \in \ \mathbb{R}^{2048},$$
$$\hat{y}_m^{\text{geno}} = h_m^{\text{geno}} W_2 + b_2 \ \in \ \mathbb{R}^3,$$

where $W_1 \in \mathbb{R}^{512\times 2048}$, $b_1 \in \mathbb{R}^{2048}$, $W_2 \in \mathbb{R}^{2048\times 3}$, and $b_2 \in \mathbb{R}^3$ are learnable parameters, and σ denotes the activation function, chosen as GELU defined in (3.2). The softmax function then converts the logits to a probability distribution over the three genotype classes

$$p(G_m = c \mid a_m^{(L)}) = \frac{\exp(\hat{y}_{m,c}^{\text{geno}})}{\sum_{k=0}^{2} \exp(\hat{y}_{m,k}^{\text{geno}})}, \quad c \in \{0, 1, 2\},$$

where c indexes the three genotype classes and $p(G_m = c \mid a_m^{(L)})$ is the predicted probability that position m has genotype c.

The training objective is to maximize the likelihood of the observed genotypes at the masked positions. Equivalently, the loss is the negative log-likelihood averaged over the masked set

$$L_{\text{geno}} = -\frac{1}{|\mathcal{M}|} \sum_{m\in\mathcal{M}} \sum_{c=0}^{2} \mathbb{1}\{G_{i,m} = c\} \log p(G_m = c \mid a_m^{(L)}), \tag{3.4}$$

where $\mathcal{M}$ is the set of masked positions, $G_{i,m}$ is the true genotype of individual i at position m, and $\mathbb{1}\{G_{i,m} = c\}$ is the indicator function that selects only the log-probability of the correct class. This is the categorical cross-entropy loss (PyTorch Contributors, 2026b).

The loss is restricted to the masked positions rather than computed over all M SNPs. If unmasked positions were included, the model could simply copy the genotype it already observes in the input, achieving a low loss without learning any contextual structure. By evaluating the loss only at positions where the input has been replaced by a mask token, the model is forced to infer the original genotype from the surrounding context, which is what drives the encoder to learn meaningful representations (Devlin et al., 2019).

3.3.2 Phenotype Prediction

For phenotype prediction, the model must produce a single prediction for the entire individual rather than a per-token prediction. A single SNP representation alone does not carry enough information to determine a phenotype, so the prediction must reflect the combined effect of all SNPs in the sequence. The encoder output $a^{(L)} \in \mathbb{R}^{M\times 512}$ is therefore aggregated by mean-pooling across the sequence dimension to obtain a single summary vector

$$\bar{a} = \frac{1}{M}\sum_{m=1}^{M} a_m^{(L)} \ \in \ \mathbb{R}^{512}.$$

Mean-pooling reduces M token representations to one vector, which necessarily discards positional information, but it provides a simple and computationally efficient aggregation that is sufficient for the classification task (Gholamalinezhad and Khosravi, 2020).

Because pooling disrupts the normalization established by the final encoder layer, a layer normalization is applied to $\bar{a}$ before the first linear mapping. The phenotype MLP is then

$$\begin{aligned}\tilde{a} &= \mathrm{LayerNorm}(\bar{a}),\\ h^{\mathrm{pheno}} &= \sigma(\tilde{a}\,W_1 + b_1),\\ \hat{y}^{\mathrm{pheno}} &= h^{\mathrm{pheno}}\,W_2 + b_2 \ \in \ \mathbb{R},\end{aligned}$$

where $W_1 \in \mathbb{R}^{512\times 2048}$, $b_1 \in \mathbb{R}^{2048}$, $W_2 \in \mathbb{R}^{2048\times 1}$, and $b_2 \in \mathbb{R}$ are learnable parameters, σ denotes the activation function, commonly chosen as GELU (3.2), and LayerNorm is defined in (3.3). The scalar output $\hat{y}^{\mathrm{pheno}}$ is a raw logit. The probability that individual i is a case is then obtained by applying the logistic (sigmoid) function

$$p_i = \sigma(\hat{y}_i^{\mathrm{pheno}}) = \frac{1}{1+\exp(-\hat{y}_i^{\mathrm{pheno}})} \ \in \ (0,\,1).$$

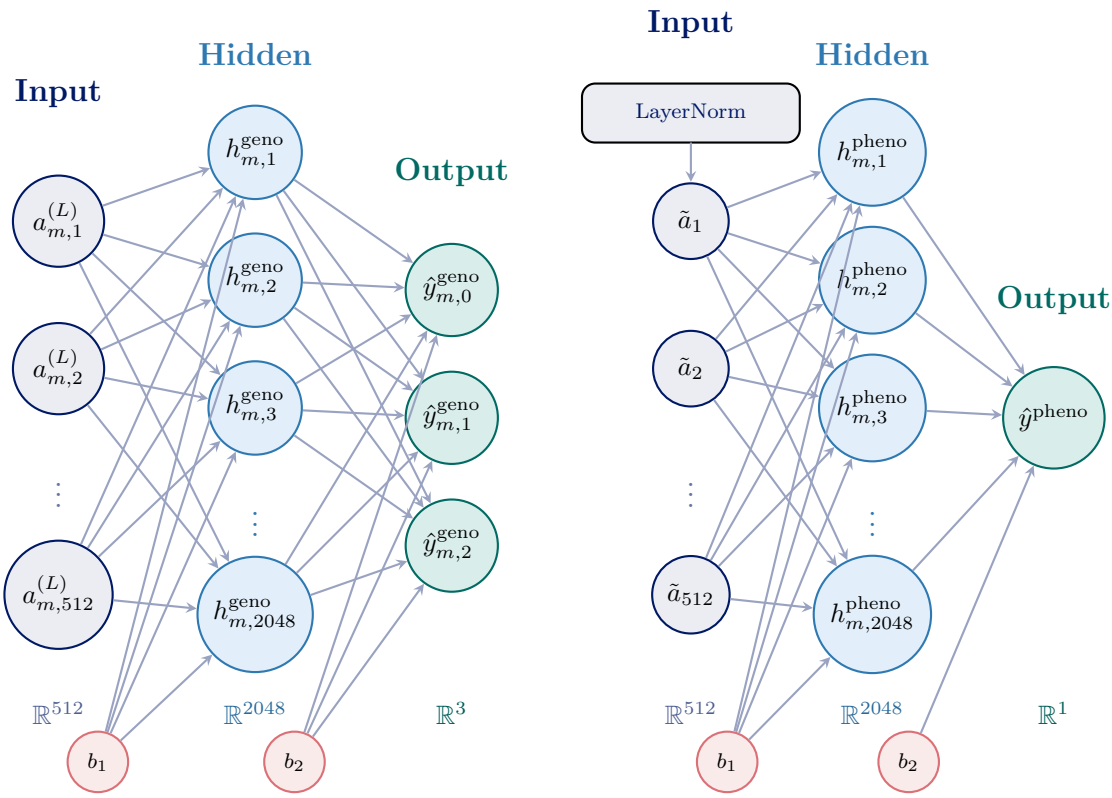
As in the genotype prediction task, the training objective is to maximize the likelihood of the observed phenotype labels. For a binary outcome $y_i \in \{0, 1\}$, the likelihood contribution of individual i is $p_i^{y_i}(1-p_i)^{1-y_i}$. Taking the negative log-likelihood and averaging over all individuals in the batch gives the binary cross-entropy loss

$$L_{\mathrm{pheno}} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log p_i + (1-y_i)\log(1-p_i)\right], \tag{3.5}$$

where N is the number of individuals in the batch (PyTorch Contributors, 2026a).

3.4 Dropout Regularization

With a large number of learnable parameters, the risk of overfitting during training is great. Dropout is a regularization technique that mitigates this by randomly setting individual activations to zero during each forward pass in training. It is applied to the input embeddings, the attention weight matrices, and the hidden-layer activations of both the feed-forward network and the prediction MLPs. The weights and biases of the model are not modified by dropout; only the intermediate representations that flow through the network are affected. By randomly removing different activations at each training step,



(a) Genotype MLP. A single-hidden-layer network that takes an individual encoder output row $a_m^{(L)} \in \mathbb{R}^{512}$ as input and produces three logits corresponding to the genotype classes $\{0, 1, 2\}$.

(b) Phenotype MLP. A single-hidden-layer network that takes the mean-pooled encoder output $\bar{a} \in \mathbb{R}^{512}$, normalized by LayerNorm, as input and produces a single logit.

Figure 3.4: Architecture of the two prediction MLPs. Both are single-hidden-layer networks with a hidden dimension of 2048 and GELU activation, but differ in how the encoder output is used, output dimensionality, and the use of layer normalization.

the model is prevented from relying on any single neuron or co-adaptation between specific neurons, and is instead forced to distribute learned information across multiple elements.

At each training step, a new random dropout mask is drawn independently, so the model sees a different subset of active neurons in every forward pass. At test time, dropout is turned off entirely: all activations are retained and the full model is used, with no averaging over multiple dropout masks (Srivastava et al., 2014a). For a given vector $x \in \mathbb{R}^d$, dropout produces

$$\text{Dropout}(x) = \frac{1}{1-p}\, x \odot m, \quad m_j \overset{\text{i.i.d.}}{\sim} \text{Bernoulli}(1-p), \quad j = 1, \ldots, d,$$

where $p \in [0, 1)$ is the dropout rate, $m \in \{0, 1\}^d$ is a binary mask of the same dimension as x with each entry sampled independently, and $\odot$ denotes element-wise multiplication. Each element of x is therefore zeroed out with probability p and retained with probability $1 - p$. The scaling factor $1/(1 - p)$ ensures that the expected value of each output element during training equals the corresponding element of x itself, so that no rescaling is needed when dropout is turned off at test time (PyTorch Contributors, 2025, Srivastava et al., 2014b). The dropout rate is set to $p = 0.1$ throughout the model (Vaswani et al., 2023).

Table 3.1: Summary of dropout throughout the model. Each row specifies the component and the computation when dropout is included.

Component	**Computation with Dropout**
Input embedding	$\text{Dropout}(E)$
Attention weights (per head)	$\text{Dropout}\left(\text{softmax}\left(\frac{Q_h K_h^\top}{\sqrt{d_k}}\right)\right) V_h$
Multi-head output	$\text{Dropout}(\text{concat}(\text{head}_1, \ldots, \text{head}_{n_h})\, W_O)$
Feed-forward network	$W_2\ \text{Dropout}\Big(\sigma(W_1\, z_m^{(\ell)} + b_1)\Big) + b_2$
Prediction MLPs hidden layer	Same as feed-forward network

3.5 Training Procedure

Training is done in two phases. First, the encoder is pre-trained using masked learning and an MLP described in subsection 3.3.1 by minimizing the loss function (3.4). In this phase, a subset of genotype tokens is masked, and the model is trained to reconstruct the original genotype values from the remaining genomic context. This encourages the encoder to learn representations that capture dependencies between SNPs without using phenotype labels.

After pre-training, the model is fine-tuned for phenotype prediction by minimizing the phenotype loss defined in subsection 3.3.2. In this phase, the pre-trained encoder is reused, and the phenotype prediction MLP is trained to map the SNP representations from the encoder to a single phenotypic outcome for each individual. Thus, the two prediction tasks are optimized sequentially.

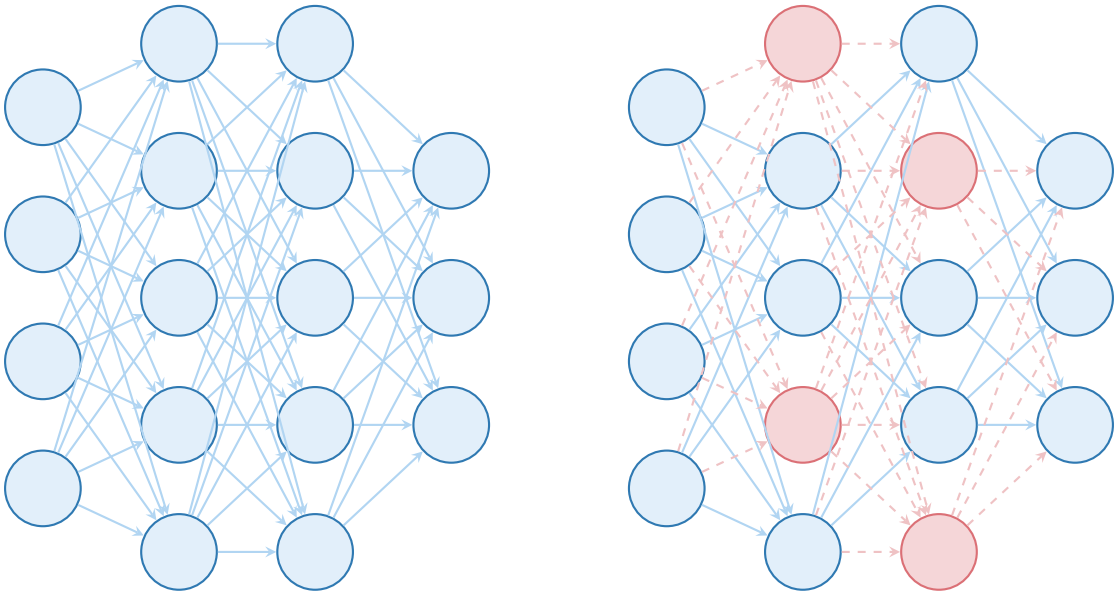
Minimizing these losses requires three steps: a partitioning of the data that allows the model to be trained, monitored, and evaluated, a method for computing the gradient of the loss with respect to every parameter in the model, and an algorithm for translating those gradients into parameter updates. The first is described in subsection 3.5.1, the second is provided by backpropagation in subsection 3.5.2, and the third by the Adaptive Moment Estimation with decoupled weight decay (AdamW) optimizer in subsection 3.5.3.

3.5.1 Data Splits and Training Iterations

A model that performs well on the data it was trained on is not necessarily useful. With enough parameters, a model can memorize the training examples entirely, known as overfitting, achieving a near-zero training loss while failing to make accurate predictions on individuals it has not seen. The ability to make accurate predictions on new, unseen data is called generalization, and it determines whether the model has learned meaningful structure rather than noise specific to the training set.

To measure generalization, the model must be evaluated on data that played no role during training. This requires partitioning the available individuals into separate sets before training begins. Each set serves a distinct purpose: one for updating the model parameters, one for monitoring progress and making training decisions, and one for the final evaluation.

The training set is the data on which the model parameters are updated. At each training step, a batch of individuals is drawn from the training set, passed through the model, and the loss is computed over these individuals. The parameters are then adjusted to reduce that loss.



(a) Before dropout. All neurons are active and all connections carry signal during the forward pass.

(b) After dropout. Red neurons have been randomly set to zero along with all their connections (dashed), forcing the remaining neurons to compensate.

Figure 3.5: Illustration of dropout regularization in an MLP. During training, each neuron is independently set to zero with probability p. The network on the right shows one possible dropout mask, where the red neurons and their edges (dashed) are dropped for this training step Srivastava et al. (2014a).

The validation set is a held-out set that the model never trains on. After each complete pass through the training set, the loss is computed on the validation set to monitor whether the model is still improving relative to previous passes, that is, whether the validation loss continues to decrease. The validation loss is also used to make training decisions: the set of model parameters that achieves the lowest validation loss is kept as the final model. Because these decisions depend on the validation set, its loss is no longer a fully unbiased estimate of generalization performance.

The test set is held out entirely from both parameter updates and training decisions. It is evaluated only once, after training is complete, on the model checkpoint that achieved the lowest validation loss. This ensures that the reported performance reflects the model's ability to generalize to data that influenced neither the learned parameters nor the choice of when to stop training. All metrics reported in this thesis are computed exclusively on the test set (Hastie et al., 2009, Sec. 7.2).

During pre-training, the N individuals are partitioned into a training set consisting of 90% of individuals and a validation set consisting of the remaining 10%. During fine-tuning, a separate partitioning is used: the individuals are split into three disjoint sets consisting of 70% for training, 10% for validation, and 20% for testing.

3.5.1.1 Epochs and mini-batches

The parameters of the model are not updated all at once. Instead, training proceeds iteratively on batches and epochs (Goodfellow et al., 2016).

A batch is a subset of B individuals drawn from the training set. At each training step,

the B individuals in the batch are passed through the model, the loss is computed over these B individuals, and the parameters are updated based on the resulting gradients. Using batches instead of the entire dataset at once serves two purposes. First, for transformer models, the full training set is typically too large to fit in memory at once. Second, computing the gradient on a batch rather than the full training set reduces the computation time per update, as the individuals within each batch can be processed in parallel on GPUs (Goodfellow et al., 2016, Sec. 8.1).

One epoch is a single complete pass through all individuals in the training set. The training set is partitioned into disjoint batches of size B, and the model processes each batch exactly once during one epoch. If the training set contains $\mathcal{I}_{\text{train}}$ individuals, then one epoch consists of $\mathcal{I}_{\text{train}}/B$ parameter updates. The training set is shuffled at the beginning of each epoch so that the partitioning into batches changes, exposing the model to different combinations of individuals across epochs. The validation and test sets are not shuffled, as they are used only for evaluation (Goodfellow et al., 2016).

3.5.2 Backpropagation

Once the model has processed a batch and computed the loss, the parameters must be adjusted to reduce that loss. The direction in which each parameter should change is determined by the gradient of the loss with respect to that parameter: if the gradient is positive, increasing the parameter would increase the loss, so the parameter should be decreased, and vice versa.

The challenge is that the model is a deep composition of functions, so the loss depends on each parameter through many intermediate computations. Computing the gradient by changing each parameter individually and observing the effect on the loss would require a separate forward pass through the model for each of the millions of parameters, which is computationally infeasible. Backpropagation addresses this by applying the chain rule systematically: starting from the loss, an error signal is propagated backward through each operation, and the gradient with respect to every parameter is extracted along the way. This requires only a single backward pass through the model (Goodfellow et al., 2016).

To illustrate the idea, consider a simplified setting: a single-hidden-layer network of the same form as the prediction MLPs defined in section 3.3. Although the full SNPbag model contains millions of parameters distributed across 16 encoder layers, the mechanism by which gradients are computed is identical. The two-layer network below captures the essential structure while keeping the algebra tractable. Given an input $x \in \mathbb{R}^{512}$, the network computes

$$\begin{aligned} h &= \text{GELU}(x\,W_1 + b_1), \\ \hat{y} &= h\,W_2 + b_2, \end{aligned}$$

where GELU is the activation function defined in (3.2). Let L denote the loss computed from $\hat{y}$ and the true target. The goal is to obtain $\partial L/\partial W_1$, $\partial L/\partial b_1$, $\partial L/\partial W_2$, and $\partial L/\partial b_2$.

To see the structure that backpropagation exploits, it helps to decompose the network into a chain of intermediate quantities. Rather than viewing the loss as one monolithic function of the parameters, introduce a name for each intermediate result:

$$\begin{aligned} u &= x\,W_1 + b_1 && \text{(pre-activation)}, \\ h &= \text{GELU}(u) && \text{(hidden activation)}, \\ \hat{y} &= h\,W_2 + b_2 && \text{(output)}, \\ L &= \ell(\hat{y}, y) && \text{(loss)}. \end{aligned}$$

Each quantity depends only on the one directly before it, forming a chain

$$x,\, W_1,\, b_1 \;\longrightarrow\; u \;\longrightarrow\; h \;\longrightarrow\; \hat{y} \;\longrightarrow\; L.$$

A parameter such as W_1 does not influence the loss directly: it first changes u, which changes h, which changes $\hat{y}$, which finally changes L. The chain rule captures exactly this: the sensitivity of L to W_1 is the product of the sensitivities along each link. For a scalar composition $L = f(g(q(\theta)))$, the chain rule states

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial g} \cdot \frac{\partial g}{\partial q} \cdot \frac{\partial q}{\partial \theta}. \tag{3.6}$$

Each factor in this product is a local derivative: it measures how sensitive one intermediate quantity is to a small change in the quantity directly before it in the chain. Backpropagation computes these local derivatives one at a time, starting from the loss and working backward, multiplying each new factor into a running product called the error signal (Goodfellow et al., 2016).

3.5.3 AdamW Optimizer

With the gradient of the loss available at every parameter, the model is trained by iteratively updating the parameters in a direction that reduces the loss. The simplest such rule is stochastic gradient descent: scale the gradient by a fixed learning rate α and subtract it from the current parameter value. While correct, this approach has two practical shortcomings in deep transformer models. First, the gradients of different parameters can vary much in scale, so a single learning rate may be far too large for some parameters and too small for others. Second, the gradient at any single training step is computed on a batch rather than the full training set, so it is noisy: the direction can fluctuate substantially from step to step, causing the optimizer to oscillate rather than converge smoothly (Kingma and Ba, 2015).

3.5.3.1 Adam

Adam maintains two running averages for each parameter: a first-moment estimate that tracks the mean gradient direction, and a second-moment estimate that tracks the gradient magnitude. At training step t, let $g_t = \nabla_\theta L(\theta_{t-1})$ denote the gradient of the loss with respect to a parameter θ, computed on the current batch via backpropagation. The two running averages are updated as

$$\begin{aligned} m_t &= \beta_1\, m_{t-1} + (1-\beta_1)\, g_t, \\ v_t &= \beta_2\, v_{t-1} + (1-\beta_2)\, g_t \odot g_t, \end{aligned}$$

where $\beta_1, \beta_2 \in [0, 1)$ are the decay rates, typically $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and $\odot$ denotes element-wise multiplication. The first moment m_t is a weighted moving average of the gradients: it carries momentum from previous gradients, smoothing the noise from the batch gradients so that the update direction reflects a running trend rather than just the current step. This momentum also accelerates convergence, as the accumulated gradients allows the optimizer to take larger effective steps in directions of consistent descent. The second moment v_t tracks the squared magnitude of each parameter's gradient, reflecting how much it has fluctuated across training steps. This gives the optimizer a sense of how noisy each parameter's gradient is: parameters with volatile gradients receive smaller, more cautious updates, while those with stable gradients are updated more aggressively.

Because m_0 and v_0 are initialized to zero, the estimates are biased toward zero in the early steps of training. To see why, consider the first step: $m_1 = (1 - \beta_1)g_1$, which is only 10% of the gradient when $\beta_1 = 0.9$. Adam corrects for this with bias-corrected estimates

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \qquad \hat{v}_t = \frac{v_t}{1 - \beta_2^t},$$

which approach m_t and v_t as t grows but counteract the under-estimation when t is small. The parameter update is then

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon},$$

where α is the learning rate, commonly set to 0.001, and ε is a small constant (typically 10^{-8}) that prevents division by zero. Dividing by $\sqrt{\hat{v}_t}$ rescales the effective step size for each parameter individually: parameters whose gradients have been consistently large receive smaller updates, preventing them from overshooting, while parameters whose gradients have been small receive proportionally larger updates, preventing them from being left behind. The Adam algorithm is summarized in 3.1.

Algorithm 3.1 Adam optimizer (Kingma and Ba, 2015).

Require: α: learning rate
Require: $\beta_1, \beta_2 \in [0, 1)$: exponential decay rates for the moment estimates
Require: ε: small constant for numerical stability
Require: $L(\theta)$: stochastic objective function with parameters θ
Require: θ_0: initial parameter vector

$m_0 \leftarrow 0$ ▷ *Initialize first moment vector*
$v_0 \leftarrow 0$ ▷ *Initialize second moment vector*
$t \leftarrow 0$ ▷ *Initialize timestep*
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta L(\theta_{t-1})$ ▷ *Gradient of the loss at timestep t*
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ ▷ *Update first moment estimate*
  $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t \odot g_t$ ▷ *Update second moment estimate*
  $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ ▷ *Bias-corrected first moment*
  $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ ▷ *Bias-corrected second moment*
  $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \varepsilon)$ ▷ *Update parameters*
return θ_t

3.5.3.2 From Adam to AdamW

A common technique for preventing the parameters from growing too large during training is weight decay, a form of regularization applied within the optimization procedure. The idea is equivalent to adding a penalty term $\frac{\lambda}{2}\|\theta\|^2$ to the loss function, where $\lambda \geq 0$ controls the penalty strength. Taking the gradient of this augmented loss introduces an additional term $\lambda\,\theta$ in the gradient, which pushes each parameter toward zero at every update step. The intuition is that large parameter values often indicate overfitting, and penalizing parameter magnitude encourages simpler, more generalizable solutions.

AdamW extends Adam by applying this weight decay directly to the parameters as a separate term in the update rule, rather than adding $\lambda\,\theta$ to the gradient before it enters the

adaptive moment estimates. This distinction matters because Adam rescales the gradient by the inverse of $\sqrt{\hat{v}_t}$: if weight decay were included in the gradient, parameters with large historical gradients would be penalized less than those with small gradients, undermining the uniformity of the regularization. By decoupling the two, AdamW ensures that all parameters are shrunk toward zero by the same proportion $\alpha\lambda$ at every step, regardless of their gradient history

$$
\begin{aligned}
m_t &= \beta_1\, m_{t-1} + (1-\beta_1)\, g_t, \\
v_t &= \beta_2\, v_{t-1} + (1-\beta_2)\, g_t \odot g_t, \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}, \\
\theta_t &= \theta_{t-1} - \alpha \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} + \lambda\, \theta_{t-1} \right),
\end{aligned}
$$

where $\lambda \geq 0$ is the weight decay coefficient. In the update rule, the term $\lambda\, \theta_{t-1}$ multiplies the current parameter value by $(1-\alpha\lambda)$ at each step, which shrinks it toward zero: after T steps without any gradient signal, a parameter would be reduced to $(1-\alpha\lambda)^T\, \theta_0$, which decays geometrically toward zero. The AdamW algorithm is summarized in 3.2.

Algorithm 3.2 AdamW optimizer (Loshchilov and Hutter, 2019).

Require: α: learning rate
Require: $\beta_1, \beta_2 \in [0, 1)$: exponential decay rates for the moment estimates
Require: ϵ: small constant for numerical stability
Require: λ: weight decay coefficient
Require: $L(\theta)$: stochastic objective function with parameters θ
Require: θ_0: initial parameter vector

$m_0 \leftarrow 0$ ▷ *Initialize first moment vector*
$v_0 \leftarrow 0$ ▷ *Initialize second moment vector*
$t \leftarrow 0$ ▷ *Initialize timestep*
while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta L(\theta_{t-1})$ ▷ *Gradient of the loss at timestep t*
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$ ▷ *Update first moment estimate*
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t \odot g_t$ ▷ *Update second moment estimate*
    $\hat{m}_t \leftarrow m_t / (1-\beta_1^t)$ ▷ *Bias-corrected first moment*
    $\hat{v}_t \leftarrow v_t / (1-\beta_2^t)$ ▷ *Bias-corrected second moment*
    $\theta_t \leftarrow \theta_{t-1} - \alpha \left(\hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) + \lambda\, \theta_{t-1} \right)$ ▷ *Update parameters with decoupled weight decay*
return θ_t

3.6 Evaluation Metrics

Training a model and evaluating its performance require different objectives. The loss function is what the optimizer minimizes, but it is not necessarily what we care about. A low cross-entropy tells us that the model assigns high probability to the correct answers, but it does not directly tell us how often the model gets the answer right or how well it distinguishes cases from controls. Evaluation metrics provide this: they quantify model performance in terms that are interpretable, comparable across models, and aligned with the use of the predictions.

The cross-entropy losses defined in (3.4) and (3.5) serve a dual role: they are the objectives that the optimizer minimizes during training, and they are also monitored on the validation and test sets to track generalization. The evaluation metrics described below, by contrast, are not used for training. They are computed only after the model has produced its predictions and serve solely to assess how well those predictions match the true values.

During pre-training, the masked genotype predictions are evaluated by the masked accuracy: the fraction of masked positions at which the predicted genotype class matches the true genotype. During fine-tuning, the phenotype predictions are evaluated by the classification accuracy at a probability threshold of 0.5 and by the area under the receiver operating characteristic curve (AUC). In all cases, the metrics are computed on the test set introduced in subsection 3.5.1.

3.6.1 Accuracy

One classification metric is accuracy: the fraction of predictions that match the true values. Given N individuals with true values $(y_1, \ldots, y_N)^\top$ and predicted labels $(\hat{y}_1, \ldots, \hat{y}_N)^\top$, accuracy is

$$\text{Accuracy} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}\{\hat{y}_i = y_i\}, \tag{3.7}$$

where $\mathbb{1}\{\cdot\}$ is the indicator function.

During pre-training, accuracy is computed over the masked positions only. For each masked position $m \in \mathcal{M}$, the predicted genotype is the class with the highest logit, $\hat{g}_m = \arg\max_{c \in \{0,1,2\}} \hat{y}^{\text{geno}}_{m,c}$, and the masked accuracy is

$$\text{Accuracy}_{\text{mask}} = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} \mathbb{1}\{\hat{g}_m = G_{i,m}\}, \tag{3.8}$$

where $G_{i,m}$ is the true genotype of individual i at position m. Unmasked positions are excluded because the model observes the true genotype at those positions in the input, so predicting them correctly requires no learned structure and would inflate the metric without reflecting any real capability (Devlin et al., 2019, Xu et al., 2025a).

During fine-tuning, the predicted phenotype label is obtained by assigning each individual to the most probable class under the fitted model. Since the predicted probability is $p_i = \sigma(\hat{y}_i^{\text{pheno}})$, this amounts to predicting case ($\hat{y}_i = 1$) when $p_i \geq 0.5$ and control ($\hat{y}_i = 0$) otherwise:

$$\hat{y}_i = \mathbb{1}\{\sigma(\hat{y}_i^{\text{pheno}}) \geq 0.5\}, \tag{3.9}$$

where $\sigma(\cdot)$ is the sigmoid function as defined in subsection 3.3.2. Accuracy is then the fraction of individuals whose predicted label matches the true phenotype.

However, accuracy can be misleading when the classes are imbalanced (Fawcett, 2006). Consider a case/control study where 5% of individuals are cases: predicting control for every individual yields 95% accuracy while being entirely uninformative. For this reason, accuracy is reported alongside the AUC described in the following subsection.

3.6.2 Area Under the Receiver Operating Characteristic Curve

Accuracy requires committing to a fixed decision threshold, yet in phenotype prediction the right threshold is rarely known in advance. The receiver operating characteristic (ROC)

curve avoids this problem by evaluating the classifier across all possible thresholds at once (Fawcett, 2006).

For a binary classifier that outputs a continuous score $s_i = \sigma(\hat{y}_i^{\text{pheno}}) \in [0, 1]$ and a threshold $\tau \in [0, 1]$, the predicted label is case ($\hat{y}_i = 1$) if $s_i \geq \tau$ and control ($\hat{y}_i = 0$) otherwise. For each τ, the classifier's performance is summarized by two conditional probabilities: the true positive rate (TPR), which is the probability of correctly predicting case among individuals who are truly cases, and the false positive rate (FPR), which is the probability of incorrectly predicting case among individuals who are truly controls

$$\text{TPR}(\tau) = P(\hat{y}_i = 1 \mid y_i = 1) = \frac{\text{TP}(\tau)}{\text{TP}(\tau) + \text{FN}(\tau)}, \tag{3.10}$$

$$\text{FPR}(\tau) = P(\hat{y}_i = 1 \mid y_i = 0) = \frac{\text{FP}(\tau)}{\text{FP}(\tau) + \text{TN}(\tau)}, \tag{3.11}$$

where $\text{TP}(\tau)$ is the number of cases correctly predicted as cases (true positives), $\text{FN}(\tau)$ is the number of cases incorrectly predicted as controls (false negatives), $\text{FP}(\tau)$ is the number of controls incorrectly predicted as cases (false positives), and $\text{TN}(\tau)$ is the number of controls correctly predicted as controls (true negatives) at threshold τ.

The ROC curve plots $\text{TPR}(\tau)$ against $\text{FPR}(\tau)$ as τ varies from 1 to 0, as depicted in Figure 3.6. At $\tau = 1$ every individual is predicted control, so $\text{TPR} = \text{FPR} = 0$ and the curve starts at the origin. As τ decreases, more individuals are classified as cases, increasing both TPR and FPR until $\tau = 0$, where every individual is predicted case and the curve reaches $(1, 1)$. A perfect classifier achieves $\text{TPR} = 1$ and $\text{FPR} = 0$ simultaneously, meaning it correctly identifies all cases while producing no false alarms among controls; this corresponds to the top-left corner of the plot, marked by the cross in Figure 3.6. A random classifier, which assigns scores independently of the true label, lies on the diagonal $\text{TPR} = \text{FPR}$.

The area under the ROC curve (AUC) summarizes the entire curve into one number, corresponding to the shaded region in Figure 3.6

$$\text{AUC} = \int_0^1 \text{TPR}\big(\text{FPR}^{-1}(u)\big)\, du. \tag{3.12}$$

An AUC of 0.5 corresponds to the diagonal in Figure 3.6 (random guessing), while 1.0 corresponds to perfect classification.

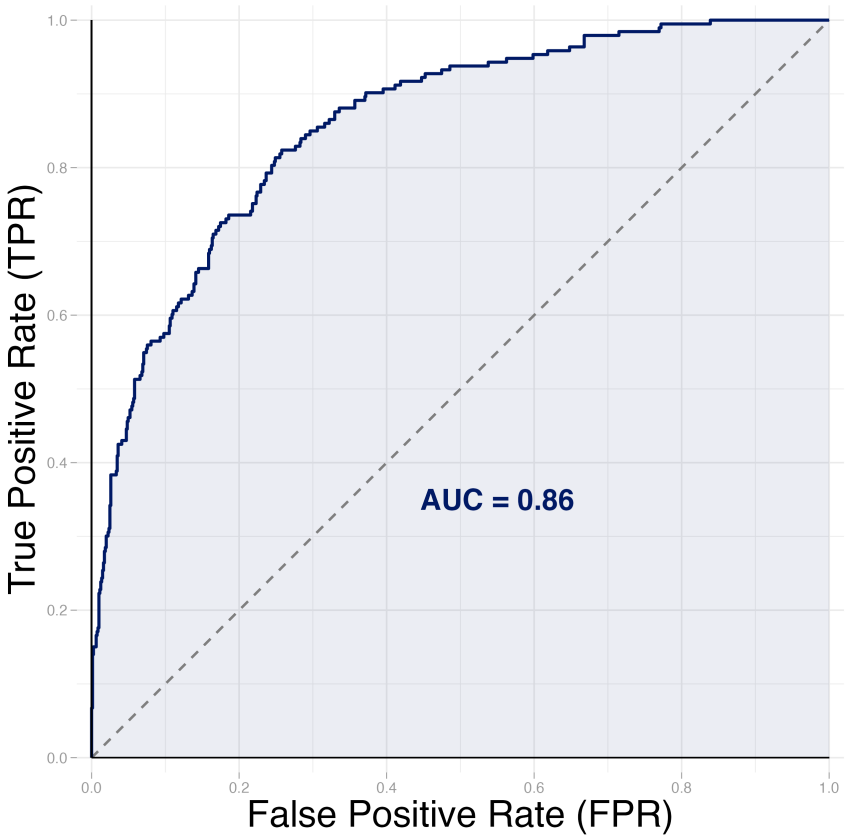


Figure 3.6: Receiver operating characteristic (ROC) curve for a logistic regression model fitted to simulated data. The data consist of $n = 1000$ observations with a binary response generated from a logistic model with three covariates, and the predicted probabilities are obtained from a logistic regression. The curve plots the true positive rate against the false positive rate as the decision threshold τ varies from 1 to 0. The shaded area corresponds to the AUC, and the dashed diagonal represents a random classifier (AUC = 0.5).

4 Phenotype Prediction on Synthetic Genomic Data

This chapter evaluates the phenotype prediction performance of SNPbag against LDpred2-auto (Privé et al., 2020). Both methods are applied to synthetic genotype data from the HAPNEST project (Wharrie et al., 2023), with phenotypes simulated under a logistic disease model. section 4.1 describes the data. section 4.2 describes how the phenotypes are simulated. section 4.3 describes how LDpred2-auto is applied. section 4.4 describes the pre-training of SNPbag. subsection 4.4.4 describes the fine-tuning of SNPbag for phenotype prediction. Finally, section 4.5 compares the two methods.

4.1 Synthetic Genotype Data

The genotype data is a synthetic dataset generated by the HAPNEST framework (Wharrie et al., 2023). HAPNEST produces individual-level genotype data by resampling phased haplotypes from reference panels (the 1000 Genomes Project and the Human Genome Diversity Project), preserving realistic linkage disequilibrium patterns and allele frequency spectra (Wharrie et al., 2023).

The dataset comprises $N = 100000$ synthetic individuals. For computational feasibility, only the first $M = 14000$ SNPs on chromosome 22 are used. The genotype data is stored in PLINK binary format (.bed, .bim, .fam), and each entry of the genotype matrix $G \in \{0, 1, 2\}^{N \times M}$ encodes the minor allele count at the corresponding locus.

The $N = 100000$ individuals are split into three non-overlapping subsets: a training set of $|\mathcal{I}_{\text{train}}| = 70000$ individuals (70%), a validation set of $|\mathcal{I}_{\text{val}}| = 10000$ individuals (10%), and a test set of $|\mathcal{I}_{\text{test}}| = 20000$ individuals (20%). This split is shared between SNPbag and LDpred2-auto so that both methods are trained and evaluated on the same individuals.

4.2 Phenotype Simulation

The phenotype is simulated directly on the $M = 14000$ SNPs from chromosome 22 because the heritability captured by this small subset is insufficient for either method to learn from the phenotypes distributed with the HAPNEST dataset. The simulation uses a logistic disease model based on Majumdar et al. (2018). The original model generates disease status from a single SNP with uniformly distributed odds ratios. Here, the model is extended to a polygenic setting: $|\mathcal{M}_c| = 500$ causal SNPs contribute simultaneously, and their log odds ratios are drawn from $\mathcal{N}(0, \sigma_\beta^2)$ rather than a uniform distribution. The logistic link and the calibration of the intercept to match a target prevalence are unchanged.

4.2.1 Causal SNPs and Effect Sizes

A set of causal SNPs $\mathcal{M}_c \subset \{1, \ldots, M\}$ with $|\mathcal{M}_c| = 500$ is sampled uniformly at random from the $M = 14000$ available SNPs. For each causal SNP $m \in \mathcal{M}_c$, a log odds ratio is drawn independently from a normal distribution

$$\beta_m \sim \mathcal{N}(0, \sigma_\beta^2), \quad m \in \mathcal{M}_c,$$

with $\sigma_\beta = 0.5$. The remaining $M - |\mathcal{M}_c|$ non-causal SNPs are assigned $\beta_m = 0$.

4.2.2 Logistic Disease Model

For individual i, the genetic component of the linear predictor is

$$\eta_i^{\text{gen}} = \sum_{m \in \mathcal{M}_c} G_{i,m}\, \beta_m,$$

where $G_{i,m} \in \{0, 1, 2\}$ is the minor allele count of individual i at SNP m. The probability that individual i is a case is given by the logistic model

$$\mathbb{P}(y_i = 1 \mid G_i) = \frac{1}{1 + \exp\left(-(\alpha + \eta_i^{\text{gen}})\right)},$$

where α is an intercept calibrated so that the overall population prevalence matches the target $K = 0.10$, i.e.

$$\frac{1}{N} \sum_{i=1}^{N} \frac{1}{1 + \exp(-(\alpha + \eta_i^{\text{gen}}))} = K.$$

The intercept α is found by univariate root-finding. The binary phenotype is then drawn as $y_i \sim \text{Bernoulli}(\mathbb{P}(y_i = 1 \mid G_i))$, producing approximately 10000 cases and 90000 controls.

The simulation outputs the binary phenotype $(y_1, \ldots, y_N) \in \{0, 1\}^N$ and the ground-truth log odds ratios for the causal SNPs.

4.3 LDpred2-auto for Phenotype Prediction

LDpred2-auto operates on GWAS summary statistics and an LD correlation matrix, as described in section 2.4. Its application to the simulated phenotype involves three steps: running a GWAS on the training set, computing the LD correlation matrix, and running the LDpred2-auto Gibbs sampler.

4.3.1 GWAS on the Training Set

A GWAS is performed on $\mathcal{I}_{\text{train}}$ using PLINK2 (Chang et al., 2015). For each of the $M = 14000$ SNPs, a logistic regression of the binary phenotype $y_i \in \{0, 1\}$ on the genotype $G_{i,m}$ is fitted without covariates, using the Firth-corrected logistic regression implemented in PLINK2. This produces an odds ratio, a log odds ratio standard error, and a p-value for each SNP. The log odds ratios are used as the marginal effect estimates $\hat{\beta}_m$ in the LDpred2 pipeline. The GWAS is performed exclusively on the training set to avoid information leakage into the summary statistics.

The effective sample size for the binary trait is computed as

$$N_{\text{eff}} = \frac{4}{1/n_{\text{case}} + 1/n_{\text{ctrl}}},$$

where n_{case} and n_{ctrl} are the number of cases and controls in the training set.

4.3.2 LD Correlation Matrix

The LD correlation matrix $R \in \mathbb{R}^{M' \times M'}$ is computed from the training genotypes at the $M' \leq M$ matched variants using the `snp_cor` function in the `bigsnpr` R package (Privé et al., 2023). A sliding window of 500 kb is used: variant pairs with physical distance

exceeding this are assumed to have zero correlation. The matrix is stored as a sparse file-backed object to reduce memory usage.

The matching step between the GWAS summary statistics and the genotype data removes ambiguous SNPs (those with complementary alleles, e.g. A/T or C/G) and any duplicates, reducing the number of variants from $M = 14000$ to $M' = 11322$.

4.3.3 Gibbs Sampler and Chain Selection

LDpred2-auto estimates the proportion of causal variants π_c and the SNP heritability h^2 directly within the Gibbs sampling procedure, as described in section 2.4. The sampler is initialised with $h^2_{\text{init}} = 0.5 \cdot (M'/M)$, scaling the initial heritability by the fraction of matched variants. Thirty independent chains are run with π_c initialised at values equally spaced on a logarithmic scale from 10^{-4} to 0.2. Each chain runs for 500 burn-in iterations followed by 500 sampling iterations.

Three stability settings are applied to prevent chain divergence on the single-chromosome setting. First, `allow_jump_sign` is set to `FALSE`, preventing effect sizes from changing sign in consecutive iterations. Second, the LD matrix is regularised with `shrink_corr` $= 0.95$, shrinking off-diagonal elements toward zero to compensate for finite-sample noise. Third, `use_MLE` is set to `FALSE`, using a more robust estimation method when GWAS power is limited (Privé et al., 2020).

Chains that diverge are discarded. Each surviving chain produces a vector of posterior mean effect sizes $\hat{\beta}^{\text{LDpred2}} \in \mathbb{R}^{M'}$, which is used to compute a PRS on the validation set. The chain with the highest validation AUC is selected, and its effect sizes are used to compute the PRS on the test set. The PRS is then used as a covariate in a logistic regression fitted on the validation set, following the standard approach in the PRS literature (Choi et al., 2020). The fitted model is applied to the test set to obtain predicted probabilities, from which accuracy and AUC are computed.

4.3.4 LDpred2-auto and Polygenic Risk Scores

LDpred2-auto is run using the `snp_ldpred2_auto` function in the `bigsnpr` R package (Privé et al., 2020), taking as input the marginal effect estimates $\hat{\beta}_m$, the LD correlation matrix R, and the effective sample size N_{eff}. The Gibbs sampler estimates the posterior mean effect sizes $\tilde{\beta}_m$, which account for LD and shrink the noisy marginal estimates toward zero.

The polygenic risk score for individual i is then computed as

$$\text{PRS}_i = \sum_{m=1}^{M'} G_{i,m}\, \tilde{\beta}_m.$$

To obtain a predicted phenotype, a logistic regression of y_i on PRS_i is fitted on the training set, and the resulting model is used to compute predicted probabilities on the test set. These probabilities are used to compute the AUC and accuracy reported in Table 4.4.

4.4 Pre-training of SNPbag

The pre-training of SNPbag follows the masked genotype modeling objective described in subsection 3.3.1. The encoder learns contextualized representations of genomic variation by reconstructing randomly masked genotype tokens from their surrounding SNPs, without access to any phenotype labels. This section describes the computational infrastructure

used for pre-training, the model architecture and hyperparameters, and the effect of training set size on pre-training performance.

4.4.1 Computational Infrastructure and Hyperparameters

Pre-training is performed on a compute node equipped with four NVIDIA B200 GPUs, each with 192 GB of memory. The model is parallelised across all four devices using PyTorch's `DataParallel` module, which replicates the model on each GPU, distributes each batch evenly across the devices, and synchronises gradients before each parameter update.

Table 4.1: Empirically estimated GPU memory consumption for different configurations of sequence length M and batch size B, using 4× NVIDIA B200 GPUs with 192 GB each. The final row exceeds the 192 GB device limit and results in an out-of-memory error.

M (SNPs)	B (batch size)	Per GPU (4 GPUs)	Est. GPU memory
7000	32	8	∼96 GB
7000	48	12	∼144 GB
14000	16	4	∼96 GB
14000	24	6	∼144 GB
27000	8	2	∼96 GB
27000	12	3	∼139 GB
50000	4	1	∼86 GB
80000	4	1	∼137 GB
116000	4	1	$>$ 192 GB

Table 4.1 shows the memory constraints during pre-training: as the number of SNPs increases, the batch size must be reduced proportionally to remain within the 192 GB memory limit of each GPU. This presents a tradeoff between genomic coverage and training speed: using more SNPs provides the model with richer genomic context but forces a smaller batch size, which increases the number of gradient updates per epoch and thus the total training time resulting in a higher computational cost.

Training on the full chromosome 22 ($M = 116524$ SNPs) is infeasible even at the minimum batch size of $B = 4$, as the estimated memory exceeds the device capacity. At $M = 80000$, training is possible but each GPU processes only a single individual per step, resulting in slow training. For this training, the configuration $M = 14000$ and $B = 24$ is used, which distributes 6 individuals per GPU per step and consumes approximately 144 GB of the available 192 GB.

Training runs for a maximum of 20 epochs, where each epoch corresponds to a complete pass through all individuals in the training set. Since the masking procedure is stochastic, a different subset of positions is masked at each epoch, so the model encounters each individual up to 20 times but never with the same input. Training is stopped early if the validation loss shows negligible improvement between consecutive epochs. The full 100000-individual run completes 20 epochs in approximately 40 hours on the four-GPU node. The complete set of hyperparameters is summarised in Table 4.2.

Table 4.2: Hyperparameters used for pre-training SNPbag.

Hyperparameter	**Value**
Embedding dimension	512
Encoder layers	16
Attention heads	16
Feed-forward dimension	2048
Mask probability	0.85
Batch size	24
Epochs	20
Learning rate	10^{-4}
Dropout probability	0.1

4.4.2 Effect of Training Set Size on Pre-training Performance

To investigate the effect of training set size on pre-training performance, the model is trained on datasets of increasing size: $N \in \{1000, 5000, 10000, 50000, 100000\}$ individuals. Each dataset is split into a training set $\mathcal{I}_{\text{train}}$ containing 90% of the individuals and a validation set $\mathcal{I}_{\text{val}}$ containing the remaining 10%, yielding the training set sizes $|\mathcal{I}_{\text{train}}| \in \{900, 4500, 9000, 45000, 90000\}$ and corresponding validation set sizes $|\mathcal{I}_{\text{val}}| \in \{100, 500, 1000, 5000, 10000\}$. For each dataset size, the same model architecture and hyperparameters are used, see Table 4.2. Both the training loss and the validation loss are recorded at each epoch. The validation loss curves are shown in Figure 4.1.

Table 4.3: Pre-training results across training set sizes. For each run, the best validation loss (Val. Loss) (lowest across all epochs), the corresponding validation accuracy (Val. Acc.), and the final training loss (Train Loss) and accuracy (Train Acc) are reported. All runs use the architecture described in subsection 4.4.1. The 50000-individual run was early stopped at epoch 12.

N	Epochs	Best Val. Loss	Val. Acc.	Train Loss	Train Acc.
1000	20	0.4714	84.18%	0.4690	84.33%
5000	20	0.4387	84.56%	0.4492	84.47%
10000	20	0.3708	85.83%	0.3721	85.86%
50000	12	0.4670	84.40%	0.4899	84.34%
100000	20	0.2544	90.19%	0.2606	89.98%

The results in Table 4.3 show a clear positive relationship between training set size and pre-training performance. Both the validation loss and the masked prediction accuracy improve monotonically as the number of training individuals increases from 1000 to 100000 with the exception of the 50000-individual run. With 1000 individuals the model achieves a validation accuracy of 84.18%, which improves to 84.56% at 5000, to 85.83% at 10000, and to 90.19% at 100000. The largest gain occurs between 10000 and 100000 individuals, where the validation loss drops from 0.3708 to 0.2544, yielding a relative reduction of 31%, suggesting that the model has not yet saturated and would likely benefit from additional training data.

The training and validation losses remain close across all dataset sizes, with no substantial gap suggesting overfitting. At 100000 individuals, the training loss 0.2606 is marginally

higher than the validation loss 0.2544.

The 50000-individual run was early stopped at epoch 12. The validation loss plateaued at 0.4881 after the first epoch, see Figure 4.1, with subsequent epochs producing improvements on the order of 10^{-4}. Given the negligible marginal improvement per epoch and the substantial computational cost of training on 50000 individuals, continuing training was not justified. The resulting validation loss 0.4670 is higher than that of the 5000-individual run 0.4387, which suggests that the optimizer had converged to a local minimum of the loss function.

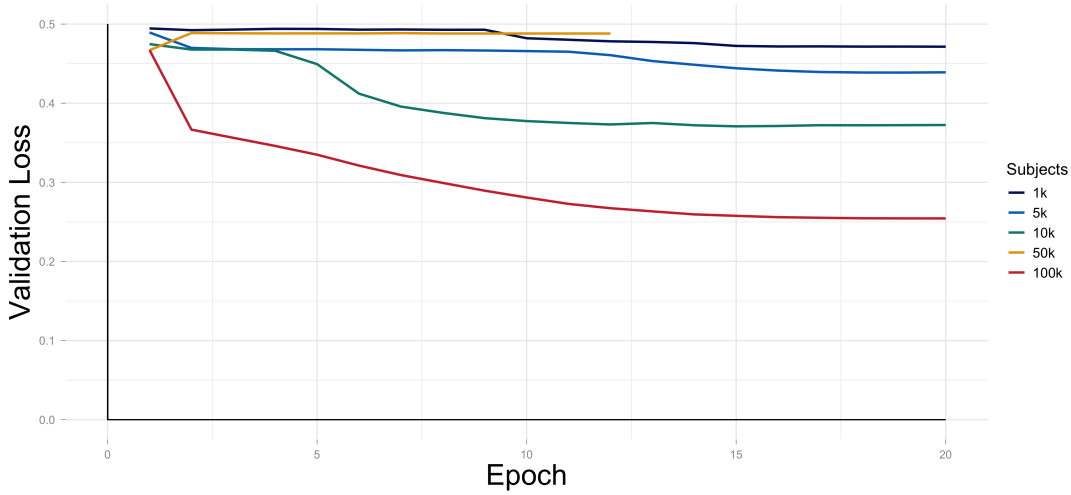


Figure 4.1: Validation loss during pre-training of SNPbag across different training set sizes. Each curve shows the masked cross-entropy loss on the validation set as a function of epoch, for training sets of 1000, 5000, 10000, 50000, and 100000 individuals. All runs use $M = 14000$ SNPs from chromosome 22, a mask probability of 0.85, and 20 epochs.

After pre-training, the SNPbag encoder has learned contextualised representations of genotype sequences from the masked genotype modelling objective without any phenotype information. Fine-tuning adapts these representations for binary phenotype prediction by replacing the masked genotype prediction head with a phenotype prediction head, as described in subsection 3.3.2.

4.4.3 Model Construction

The fine-tuning model is constructed by loading the pre-trained encoder parameters with the lowest validation loss from the 100000-individual run as the starting point of the prediction model.. The encoder and embedding architecture is identical to the pre-training configuration described in Table 4.2. The masked genotype prediction head is discarded and replaced by a phenotype head, as described in subsection 3.3.2.

Since the phenotype is defined for an entire individual rather than per-SNP, the encoder outputs are aggregated by mean pooling into a single vector per individual, which is passed to the phenotype head to obtain a predicted logit, as described in subsection 3.3.2.

4.4.4 Training Procedure

Fine-tuning uses the same 70/10/20 data split as LDpred2-auto, ensuring both methods are evaluated on identical individuals.

Due to a limited computational budget, the pre-trained encoder parameters are frozen during fine-tuning. This means that only the phenotype head is trained, while the encoder and embedding weights remain fixed at the values learned during pre-training. Freezing the encoder substantially decreases both the memory footprint and the time required per training step. The trade-off is that the encoder cannot adapt its representations to the phenotype prediction task, so the model relies entirely on the genomic context already captured during masked genotype pre-training.

Training was done for a single epoch with batch size 144, distributed across 8 GPUs, and the resulting model is evaluated on the test set using accuracy and AUC.

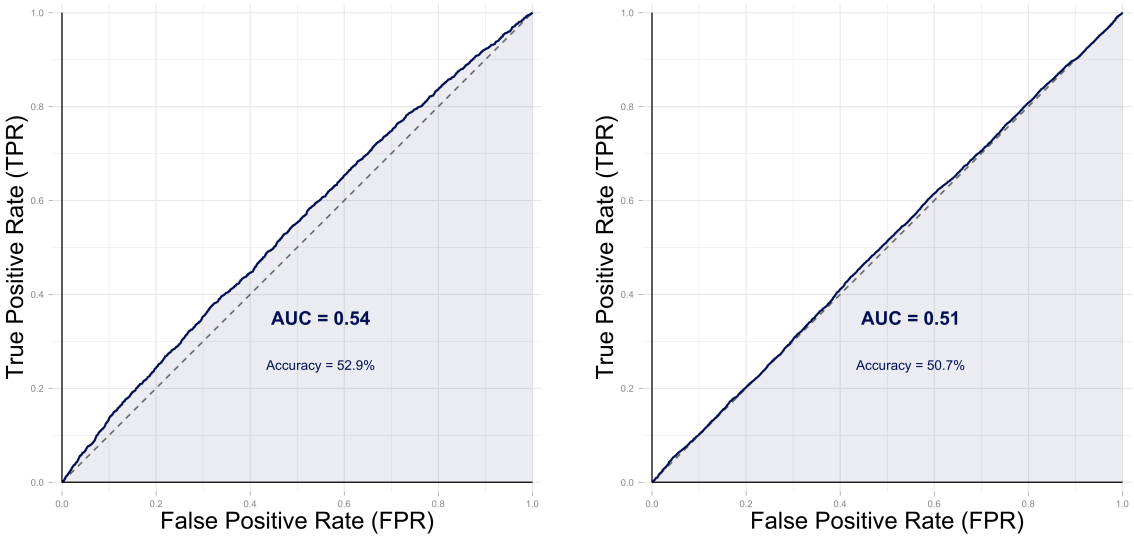
4.5 Comparison of SNPbag and LDpred2-auto

Both methods are evaluated on the same test set $\mathcal{I}_{\text{test}}$ of 20000 individuals using two metrics: classification accuracy at a threshold tuned on the validation set, and the area under the receiver operating characteristic curve (AUC). The results are summarised in Table 4.4 and the ROC curves are shown in Figure 4.2.

Table 4.4: Test set performance of LDpred2-auto and SNPbag on the simulated binary phenotype. The phenotype is simulated with the logistic disease model (Majumdar et al., 2018) with $|\mathcal{M}_c| = 500$ causal SNPs, $\sigma_\beta = 0.5$, and a target prevalence of $K = 0.10$. Both methods are trained on 70 000 individuals, validated on 10 000 individuals, and tested on 20 000 individuals.

Method	**Accuracy**	**AUC**
LDpred2-auto	`0.507`	`0.510`
SNPbag	`0.529`	`0.540`

The SNPbag achieved an accuracy of 52.9%, with an AUC of 0.54, only slightly above the random baseline of 0.5, which confirms that the model has minimal discriminative power. LDpred2-auto similarly shows a near-random AUC of 0.51. Neither method achieved meaningful phenotype prediction in this setting. This is likely due to a combination of factors: SNPbag was limited to a single epoch with a frozen encoder, LDpred2-auto operated on only 14000 SNPs from a single chromosome, and genotype-based prediction can at most explain the heritability h^2 of the trait, meaning that a substantial fraction of the phenotypic variation is driven by environmental factors that no genotype model can capture.



(a) SNPbag (AUC = 0.54). **(b)** LDpred2-auto (AUC = 0.51).

Figure 4.2: ROC curves on the test set for SNPbag and LDpred2-auto on the simulated binary phenotype described in Table 4.4. The shaded area corresponds to the AUC, and the dashed line represents a random classifier (AUC = 0.5). Both curves lie close to the diagonal, indicating that neither method achieves meaningful discriminative power in this setting.

5 Discussion

Neither SNPbag nor LDpred2-auto achieved meaningful discriminative performance on the simulated phenotype, with AUCs of 0.54 and 0.51 respectively. The remainder of this chapter examines the limitations that contributed to this outcome and identifies directions for future work.

The most significant limitation is the restricted genomic coverage. The model was trained on only 14000 SNPs from a single chromosome, a constraint imposed entirely by GPU memory. Processing the full chromosome 22 (~ 116000 SNPs) would exceed the memory of a single NVIDIA B200 GPU even at the smallest batch size, and processing all 22 autosomes would require fundamentally different parallelisation strategies. At the same time, the pre-training scaling experiment showed that validation loss decreased monotonically as the training set grew from 1000 to 100000 individuals, with the largest relative improvement occurring between 10000 and 100000. This suggests that the model has not yet saturated and would benefit from both longer sequences and more training individuals.

All experiments were conducted on synthetic data generated by the HAPNEST framework, not on real genotype data from a clinical cohort. While HAPNEST preserves realistic linkage disequilibrium patterns and allele frequency spectra, synthetic data may not fully capture the complexity of real human genomes, including population substructure, rare variants, and genotyping errors. The phenotype simulation used in this thesis assumes a simple additive logistic model with a known set of causal variants, which may be more favourable to both methods than the complex genetic architectures underlying real diseases. Whether the pre-trained representations would transfer to real data remains an open question.

Several architectural and methodological choices could be explored in future work. SNPbag masks 85% of genotype tokens during pre-training, far higher than the 15% used in the original BERT model (Devlin et al., 2019). This aggressive masking forces the encoder to reconstruct most of the sequence from very limited context, which may strengthen the learned representations but could also make the pre-training task unnecessarily difficult. The effect of the masking rate on downstream phenotype prediction has not been investigated. Similarly, the GELU activation function was inherited from the original BERT architecture without exploration of alternatives such as SiLU or SwiGLU, which have shown improvements in large language models. The prediction head is a simple MLP applied to mean-pooled encoder output, and alternative aggregation or prediction strategies could also be considered. Each of these would require minimal code changes and could be evaluated as straightforward ablation studies.

Finally, it is important to consider whether potential gains justify the computational cost. Larger models and longer sequences may improve prediction, but if the improvement is marginal or inconsistent across traits, the added environmental and financial cost may not be worthwhile. The results of this thesis suggest that the transformer architecture is viable for learning genomic representations, the pre-training task achieved 90% masked accuracy, but that the current bottleneck is scale, not architectural design.

6 Conclusion

This thesis developed SNPbag, a transformer-based model for phenotype prediction from genotype data, and compared it against LDpred2-auto on simulated binary phenotypes. The evaluation used 100000 synthetic individuals from the HAPNEST project with 14000 SNPs from chromosome 22 and phenotypes simulated under the logistic disease model of Majumdar et al. (2018).

The pre-training phase demonstrated that transformer-based self-supervised learning can capture meaningful genomic structure. Masked genotype prediction accuracy improved consistently from 84.18% to 90.19% as the training set grew from 1000 to 100000 individuals, with training and validation losses remaining close throughout, indicating no overfitting. The continued improvement at 100000 individuals suggests the model has not yet saturated.

Despite effective pre-training, neither method achieved strong phenotype prediction. SNPbag obtained an AUC of 0.54 and LDpred2-auto an AUC of 0.51, both only marginally above the random baseline of 0.5. Several factors explain this. SNPbag was fine-tuned for a single epoch with a frozen encoder, meaning the representations could not adapt to the phenotype prediction task and the MLP had to learn the entire mapping from fixed features in one pass. LDpred2-auto was designed for genome-wide summary statistics but operated on only 14000 SNPs from a single chromosome, capturing a small fraction of the genetic variation it would normally leverage. More fundamentally, genotype-based prediction is bounded by the heritability h^2 of the trait, so a substantial fraction of the phenotypic variation is driven by environmental factors that no genotype model can capture. The combination of limited genomic coverage, restricted fine-tuning, and the inherent heritability ceiling makes the near-random performance unsurprising.

The key conclusion is that the bottleneck is scale, not architecture. The pre-training results show that the transformer learns useful genomic representations, and SNPbag achieved a marginally higher AUC than LDpred2-auto despite operating under severe constraints. With greater computational resources, future work could extend the model to longer sequences, include more training individuals, and fine-tune the full encoder, each of which addresses one of the limitations identified above.

Bibliography

(2023). Homologous chromosome.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization.

Boué, L. (2025). Deep learning for pedestrians: backpropagation in transformers.

Chang, C. C., Chow, C. C., Tellier, L. C. A. M., Vattikuti, S., Purcell, S. M., and Lee, J. J. (2015). Second-generation PLINK: rising to the challenge of larger and richer datasets. *GigaScience*, 4(1):7.

Choi, S. W., Mak, T. S.-H., and O'Reilly, P. F. (2020). Tutorial: a guide to performing polygenic risk score analyses. *Nat. Protoc.*, 15(9):2759–2772.

Cordell, H. J. (2009). Detecting gene–gene interactions that underlie human diseases. *Nature Reviews Genetics*, 10(6):392–404.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT)*, pages 4171–4186.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognit. Lett.*, 27(8):861–874.

for Genetic, G. A. T. N. Y.-M.-A. C. and Services., N. S. (2009). *Understanding Genetics: A New York, Mid-Atlantic Guide for Patients and Health Professionals.* https://www.ncbi.nlm.nih.gov/books/NBK115568/.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. (2013). *Bayesian Data Analysis*. Chapman and Hall/CRC, 3rd edition.

Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741.

Gholamalinezhad, H. and Khosravi, H. (2020). Pooling methods in deep neural networks, a review. *arXiv preprint arXiv:2009.07485*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Graffelman, J. and Camarena, J. M. (2007). Graphical tests for hardy-weinberg equilibrium based on the ternary plot. *Human Heredity*, 65(2):77–84.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning*. Springer, 2 edition.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition.

Hendrycks, D. and Gimpel, K. (2016). Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*.

Majumdar, A., Haldar, T., Bhattacharya, S., and Witte, J. S. (2018). An efficient bayesian meta-analysis approach for studying cross-phenotype genetic associations. *PLoS Genet.*, 14(2):e1007139.

Marees, A. T., de Kluiver, H., Stringer, S., Vorspan, F., Curis, E., Marie-Claire, C., and Derks, E. M. (2018). A tutorial on conducting genome-wide association studies: Quality control and statistical analysis. *Int. J. Methods Psychiatr. Res.*, 27(2):e1608.

Privé, F., Albiñana, C., Arbel, J., Pasaniuc, B., and Vilhjálmsson, B. J. (2023). Inferring disease architecture and predictive ability with LDpred2-auto. *The American Journal of Human Genetics*, 110:2042–2055.

Privé, F., Arbel, J., and Vilhjálmsson, B. J. (2020). LDpred2: better, faster, stronger. *Bioinformatics*, 36(22-23):5424–5431.

PyTorch Contributors (2025). torch.nn.dropout. https://docs.pytorch.org/docs/2.12/generated/torch.nn.Dropout.html. Accessed: 2026-05-20.

PyTorch Contributors (2026a). Bcewithlogitsloss. https://docs.pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html. Accessed: 2026-05-20.

PyTorch Contributors (2026b). Crossentropyloss. https://docs.pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html. Accessed: 2026-05-20.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014a). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014b). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Torkamani, A., Wineinger, N. E., and Topol, E. J. (2018). The personal and clinical utility of polygenic risk scores. *Nature Reviews Genetics*, 19(9):581–590.

Uffelmann, E., Huang, Q. Q., Munung, N. S., de Vries, J., Okada, Y., Martin, A. R., Martin, H. C., Lappalainen, T., and Posthuma, D. (2021). Genome-wide association studies. *Nat. Rev. Methods Primers*, 1(1).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2023). Attention is all you need.

Visscher, P. M., Wray, N. R., Zhang, Q., Sklar, P., McCarthy, M. I., Brown, M. A., and Yang, J. (2017). 10 years of GWAS discovery: Biology, function, and translation. *The American Journal of Human Genetics*, 101(1):5–22.

Wharrie, S., Yang, Z., Raj, V., Monti, R., Gupta, R., Wang, Y., Martin, A. R., O'Connor, L. J., Kaski, S., Marttinen, P., Palamara, P. F., Lippert, C., and Ganna, A. (2023). HAPNEST: efficient, large-scale generation and evaluation of synthetic datasets for genotypes and phenotypes. *Bioinformatics*, 39(9):btad535.

Xu, A. G., Xu, Y., Xing, Y., Luo, P., Yang, J., Bai, Y., and Tang, K. (2025a). Towards a universal foundation model for biobank-scale human genome variation.

Xu, A. G., Xu, Y., Xing, Y., Luo, P., Yang, J., Bai, Y., and Tang, K. (2025b). Towards a universal foundation model for biobank-scale human genome variation.

Zaitlen, N. and Kraft, P. (2012). Heritability in the genome-wide association era. *Hum. Genet.*, 131(10):1655–1664.