Tree sequences as a general-purpose tool for population genetic inference

Logan S. Whitehouse[1*], Dylan Ray[1*], and Daniel R. Schrider[1@]

Affiliations: [1]Department of Genetics, University of North Carolina, Chapel Hill, North Carolina, USA

Address: [1]120 Mason Farm Rd, Chapel Hill, NC 27514

[@]Corresponding author

[*]Contributed equally to this work

E-mail: drs@unc.edu

Running head: Population genetic inference with tree sequences

Keywords: tree sequences, machine learning, population genetics

ABSTRACT

As population genetics data increases in size new methods have been developed to store genetic information in efficient ways, such as tree sequences. These data structures are computationally and storage efficient, but are not interchangeable with existing data structures used for many population genetic inference methodologies such as the use of convolutional neural networks (CNNs) applied to population genetic alignments. To better utilize these new data structures we propose and implement a graph convolutional network (GCN) to directly learn from tree sequence topology and node data, allowing for the use of neural network applications without an intermediate step of converting tree sequences to population genetic alignment format. We then compare our approach to standard CNN approaches on a set of previously defined benchmarking tasks including recombination rate estimation, positive selection detection, introgression detection, and demographic model parameter inference. We show that tree sequences can be directly learned from using a GCN approach and can be used to perform well on these common population genetics inference tasks with accuracies matching or exceeding that of a less efficient CNN-based method. As tree sequences become more widely used in population genetics research we foresee developments and optimizations of this work to provide a foundation for population genetics inference moving forward.

INTRODUCTION

Recent decades have seen dramatic growth in the volume, scale, and diversity of population genetic datasets. This explosion in the amount of data has coincided with the development of an expansive set of statistical methods designed to make more accurate and detailed evolutionary inferences from this glut of data. Initial methods often relied on examining the values of statistics

41 that summarize relevant aspects of genetic diversity (Watterson 1975; Nei and Li 1979; Tajima
42 1989; Fu and Li 1993). For example, statistics quantifying information about the presence of
43 high-frequency derived alleles may be used to identify recent positive selection, which produces
44 an excess of high-frequency derived alleles via genetic hitchhiking (Fay and Wu 2000). In
45 addition to statistics that summarize the distribution of allele frequencies in a population,
46 researchers have designed statistics to capture other properties of genetic variation as well, such
47 as the density of polymorphism (Nei and Li 1979; Nei and Tajima 1981), patterns of haplotypic
48 diversity (Hudson *et al.* 1994; Sabeti *et al.* 2002; Voight *et al.* 2006; Garud *et al.* 2015), and the
49 extent of linkage disequilibrium (Kelly 1997; Kim and Nielsen 2004). Because such summary
50 statistics necessarily result in the loss of potentially valuable information by representing
51 population genomic diversity by a single number, more recent approaches, such as approximate
52 Bayesian computation (ABC) sought to combine multiple summaries into a higher-dimensional
53 representation of the data that could then be used for inference (Tavare *et al.* 1997; Pritchard *et*
54 *al.* 1999; Beaumont *et al.* 2002). ABC methods work by comparing large sets of simulations,
55 each summarized by a vector of statistics, to identify the evolutionary model and/or parameter
56 combination that best matches the observed data. Importantly, the summary statistic vector can
57 capture a variety of properties of genetic variation, that together may be informative about the
58 evolutionary models/parameters being considered.
59 More recently, researchers have begun developing supervised machine learning (SML)
60 methods for population genetic inference (Schrider and Kern 2018; Korfmann *et al.* 2023a).
61 Unlike traditional applications of supervised learning, these approaches use simulated data to
62 train and test methods before applying them to empirical datasets. This reliance on simulated
63 training data makes SML approaches to population genetic inference similar in principle to ABC,

although these methods often have greater accuracy and computational efficiency (Pudlo *et al.* 2016; Raynal *et al.* 2019). SML methods have been applied to a variety of population genetic tasks including detecting positive selection (Pavlidis *et al.* 2010; Lin *et al.* 2011; Ronen *et al.* 2013; Pybus *et al.* 2015; Kern and Schrider 2018; Mughal and DeGiorgio 2019; Mughal *et al.* 2020; Hejase *et al.* 2022; Whitehouse and Schrider 2022; Arnab *et al.* 2022), demographic parameter estimation (Sheehan and Song 2016; Wang *et al.* 2021) and model inference (Sanchez *et al.* 2021), inferring recombination rates (Gao *et al.* 2016; Adrion *et al.* 2020), detecting introgression (Schrider *et al.* 2018; Gower *et al.* 2021; Ray *et al.* 2023), and numerous other applications (Battey *et al.* 2020, 2021; Yelmen *et al.* 2021; Smith *et al.* 2023; Booker *et al.* 2023).

The initial applications of SML to population genetic problems took as their input a vector of summary statistics, and yielded sizable accuracy gains on several tasks (reviewed in Schrider and Kern 2018). Because representing genetic diversity within a population by a set of statistics may still result in valuable information being lost, researchers have more recently begun to experiment with deep learning methods that are capable of directly examining a population genetic alignment as input and learning their own set of features to extract from these data (Chan *et al.* 2018; Flagel *et al.* 2019). Although deep learning has allowed for a multitude of significant advances in population genetics (reviewed in Korfmann *et al.* 2023a), one important drawback of this approach is that population genetic alignments are an inefficient format for representing data (Kelleher *et al.* 2019). Because of the large amount of information that must be stored in onboard GPU memory, examining large genomic regions or entire chromosomes and/or large sample sizes is a significant computational challenge for deep learning methods that take alignments as their input.

87 One approach to alleviate this issue is to store genetic information as tree sequences: the
88 sequence along the chromosome of genealogies describing the shared evolutionary history of the
89 sequenced genomes (Kelleher *et al.* 2016), with adjacent genealogies in this sequence separated
90 by the breakpoints of recombination events that resulted in a change to the sample's genealogy
91 (Wiuf and Hein 1999). Tree sequences can be used to concisely store all of the information
92 present in a population genetic alignment, and this innovation has led to dramatic improvements
93 in simulation efficiency (Kelleher *et al.* 2018; Haller and Messer 2022). Relatedly, tree
94 sequences can be used as a hyper-efficient form of lossless compression for population genetic
95 data (Kelleher *et al.* 2019), and can in principle describe the near-complete evolutionary history
96 of a sample. These advantages have motivated the development of methods to use sequence data
97 to infer tree sequences (Kelleher *et al.* 2019; Speidel *et al.* 2019; Zhang *et al.* 2023) or ancestral
98 recombination graphs (ARGs; Rasmussen *et al.* 2014; Mahmoudi *et al.* 2022), a similar but more
99 complete representation of a sample's history.
100 In addition to inferring tree sequences from data, there is growing interest in conducting
101 inference on the tree sequences themselves. For example, tree sequences have been used to
102 efficiently compute commonly used summary statistics with dramatic improvements in speed
103 and memory usage (Ralph *et al.* 2020). Researchers have also begun developing methods using
104 tree sequences or summaries thereof as input to machine learning methods for evolutionary
105 inference (Hejase *et al.* 2022; Pearson and Durbin 2023; Korfmann *et al.* 2023b). Given that
106 when applied to real datasets such methods will be run on tree sequences that are inferred with a
107 considerable amount of error (Zhang *et al.* 2023) one concern may be that tree-sequence methods
108 using erroneous input information may produce less satisfactory results than methods directly
109 acting on the input sequence data. However despite this potential drawback SML methods that

are trained to handle potential errors in their tree-sequence input could potentially circumvent this problem (Mo and Siepel 2023), resulting in successful use of SML on inferred tree sequences.

With these potential strengths and weaknesses of tree sequence-based inference in mind, we sought to assess whether deep learning approaches trained on tree sequences are likely to add value in comparison to methods acting directly on population genetic alignments. We address this question by assessing the performance of graph convolutional networks (GCNs), a generalization of convolutional neural networks (CNNs) to handle graphical input data. We train GCNs to solve common tasks in population genetic inference: detecting loci that have experienced introgression between closely related species, identifying selective sweeps, estimating recombination rates, and inferring demographic parameters (the same tasks examined previously by Flagel *et al.* (2019)). Encouragingly, we find that GCNs trained to use inferred tree sequences yield equivalent or better accuracy to those trained on population genetic alignments. We close with a discussion of the implications of our GCN's performance for future population genetic research leveraging this increasingly commonly used representation of evolutionary histories and genetic variation data.

RESULTS

Our goal is to assess the general effectiveness of the approach of performing population genetic inference from inferred tree sequences. To this end, we evaluate the performance of graph convolutional networks (GCNs), a type of deep neural network that takes graphical data as its input, when applied to inferred tree sequences. We gauge the success of GCNs by comparing their performance to that of CNNs trained on image representations of the input population

genetic alignment data—this comparison is an informative measuring stick given that the CNN-image approach has been shown to be highly effective on a number of population genetic problems, as discussed above. We focus our assessment on the four tasks examined by Flagel *et al.* (2019): estimating historical recombination rates, detecting and classifying selective sweeps, detecting the presence and directionality of introgression, and inferring demographic parameters. We begin with an overview of GCNs and how they differ from CNNs as well as their application to tree sequences before addressing the two approaches' performance on each problem in turn.

Graph convolutional networks on tree sequences

Convolutional neural networks (CNNs) have proved remarkably effective in a variety of fields (e.g. Hochreiter and Schmidhuber 1997; LeCun *et al.* 1998; Erhan *et al.* 2013; and reviewed in Gu *et al.* 2018; Li *et al.* 2022), and in recent years have been successfully applied to a number of problems in population genetics (Korfmann *et al.* 2023a). These networks work by extracting features from images, or other structured data that can be represented as tensors of two or more dimensions such as audio or video data, using convolutional filters. By using a series of convolution filter layers, often combined with pooling layers that downsample the data, CNNs are able to extract features within the data that are pertinent to the target task. These features are typically then passed as input into a one or more fully connected layers that then emit the final prediction: a vector of values corresponding to either to class membership probabilities in the case of classification (e.g. with each class representing a different possible object present in the image), or to estimated parameter/attribute values in the case of regression. The convolutional filters in a CNN, each of which is smaller than the input to which they are applied, work by striding across the input image and at each step performing a transformation of the portion of the

input image that is currently "covered" by the filter (LeCun *et al.* 1989). The transformed value is a function of the pixels in the input image and also the values of the weights of the filter (specifically, the grand sum of the Hadamard product between the filter and the corresponding portion of the input). The filter then moves to another location of the input and repeats the process, until a filtered version of the image has been produced. During training, the weights of the convolutional filters are adjusted to minimize error on the task at hand, thereby producing a set of filters that are capable of extracting low-level structural features (e.g. edges in an image) from the input image that are informative for downstream classification/regression by the rest of the neural network. When multiple layers of convolutions are stacked on top of each other, with the output from one layer passed to the next, features about higher level structures present in the input (e.g. shapes in the image) can be extracted.

GCNs work similarly to CNNs in that they extract features from an input using parameterized filters to convolve over the input space. As the name implies however, rather than an image or "image-like" data structure GCN input consists of a graph. This graph is represented as a pair of matrices: 1) an adjacency matrix describing the structure and connectedness of the nodes in the graph, and 2) a feature matrix containing a vector of attributes for each node in the graph. When centered on a focal node, convolutions in GCNs incorporate information from each of its neighbors, with iterative convolutions passing information from further nodes through information sharing and, in the case of graph attention networks (GATs), attention. GCNs are thus a generalization of CNNs, with the latter being a special case where each pixel in the input corresponds to a node within a lattice graph structure. The design of GCNs allows them to learn from explicit connectedness in a graph to make inferences based on topology and features. Although a relatively new form of neural networks, GCNs have been shown to have excellent

accuracy on tasks where graph data is relevant and have become a widely-researched type of neural network (reviewed in Zhou *et al.* 2020), however they have had only limited use in population genetics applications (Korfmann *et al.* 2023a; b).

Because genealogies are a type of graph, the trees in a tree sequence are a fitting input for a GCN. However, in order to extract the most information possible for inference the spatial relationship between trees must be considered as well. To accomplish this we built a hybrid GCN-RNN architecture to learn both the graph structure of each tree in a sequence as well as the spatial relationship of trees along a chromosome. Each tree is passed through six graph convolution layers which extract information from an individual tree in the sequence comprising of node features (node age, number of mutations, population label; see Methods for more details) and the connectedness of the graph through adjacency matrix (represented by edge indices). After graph convolution, we pass the node features through two RNNs (recurrent neural networks), the first of which compresses the features from each tree into a single feature vector and the second compresses the features from each sequence of trees into a single vector. This vector is then used as input to a fully-connected network (MLP) that is used to optimize output for a given task. This combination of models optimizing for a single task results in an architecture able to learn both the spatial relatedness of nodes within a tree and their relevance as well as the larger-scale structured information of the tree sequence as a whole. This architecture is displayed in Figure 1.

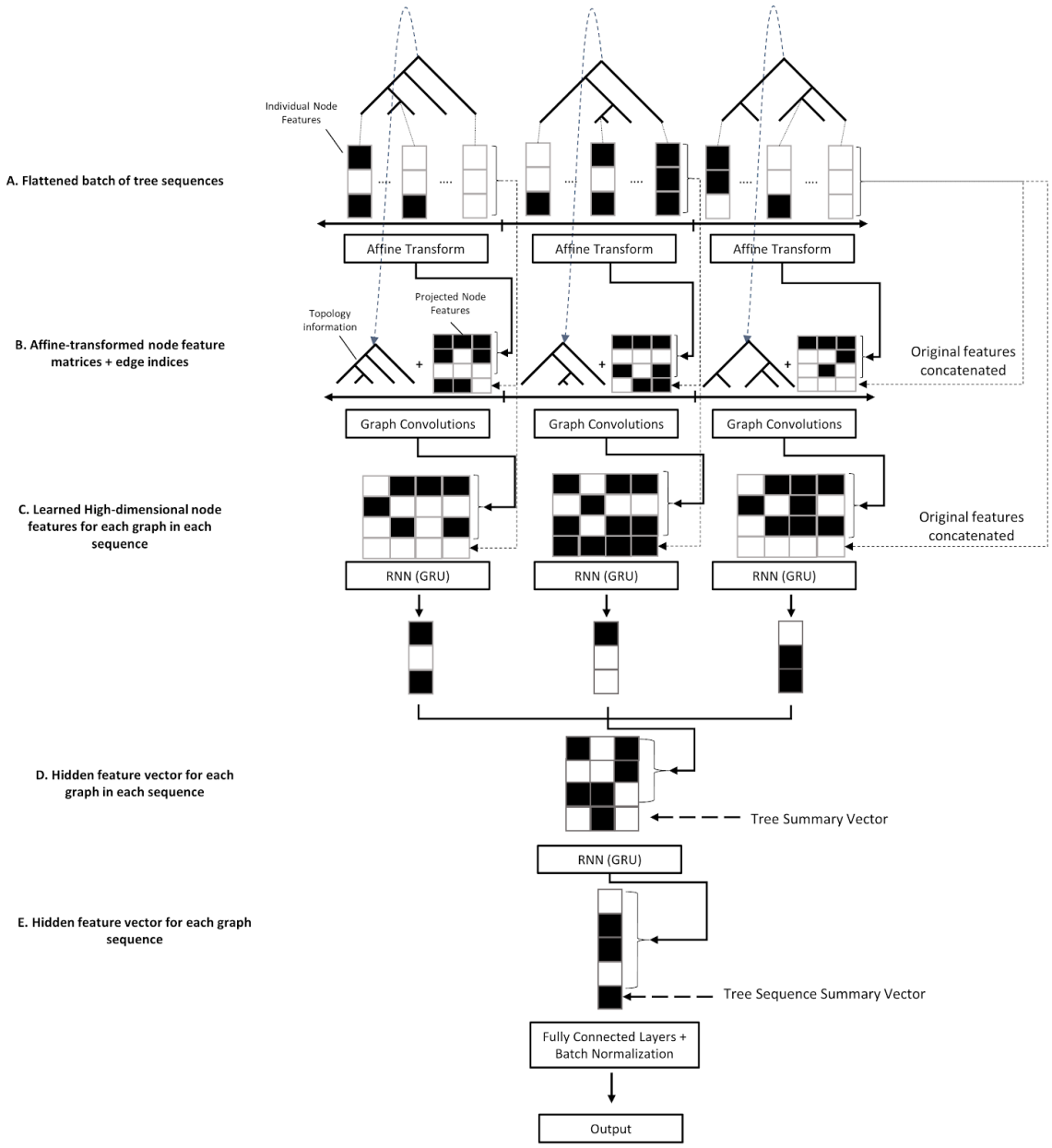


Figure 1. Overview of the GCN architecture. A) Each tree in the input tree sequence contains n nodes each summarized by f features, and is thus represented by an $n \times f$ node feature matrix per-tree, which is affine-transformed in the "node embedding" layer of the neural network (Table 1). B) For each tree, the affine-transformed node feature matrix is concatenated to a vector of edge indices specifying the tree topology, and also to the original feature matrix (i.e. a "skip connection"). This concatenated matrix is then input to multiple graph convolutional layers with skip connections. C) For each tree, the features from graph convolution layers are then fed as input to a GRU, resulting in a compressed, learned vector representation of the tree and its

features. D) These compressed vector representations are concatenated to a "tree summary vector" that encodes summaries of the size and shape of the tree, as well as the size and location of the chromosomal segment corresponding to the tree. The resulting vectors for all trees are then concatenated together into one matrix corresponding to the entire tree sequence, and passed through another GRU layer that produces a flattened vector representation of the entire tree sequence. E) This vector is then concatenated to a "tree-sequence summary vector" containing information about the distribution of values in the original tree summary vectors and the number of trees in the original sequence (which may be less than the padded length of this vector). This vector is then passed through multiple fully-connected layers culminating in a final output layer tailored to the target task. Throughout the diagram, solid lines with arrows show the flow of information through layers that transform the input, while dashed lines with arrows show the flow of information that has been unaltered by any preceding layers in the network.

GCNs can leverage tree-sequence data to accurately estimate recombination rates

Estimating historical recombination rates is a deeply important task to population genetics as the recombination rate modulates key evolutionary processes and also affects our ability to make inferences about these processes. For example, when considering the effects of selection, the recombination rate between loci influences the degree of selective interference between them (Hill and Robertson 1966). Moreover, in the case of positive selection, the signatures of selection are much more difficult to detect in the presence of high rates of recombination due to the rapid exchange of alleles, which dampens the hitchhiking effect by preserving more genetic diversity in the vicinity of the selected site (Shriner *et al.* 2003). Because the recombination rate has a direct influence on the correlation between alleles in a sample of genomes, population genomic data can in principle be used to get an accurate estimate of the recombination rate landscape across the genome (Hudson 2001; Auton and McVean 2007; Chan *et al.* 2012; Gao *et al.* 2016; Adrion *et al.* 2020).

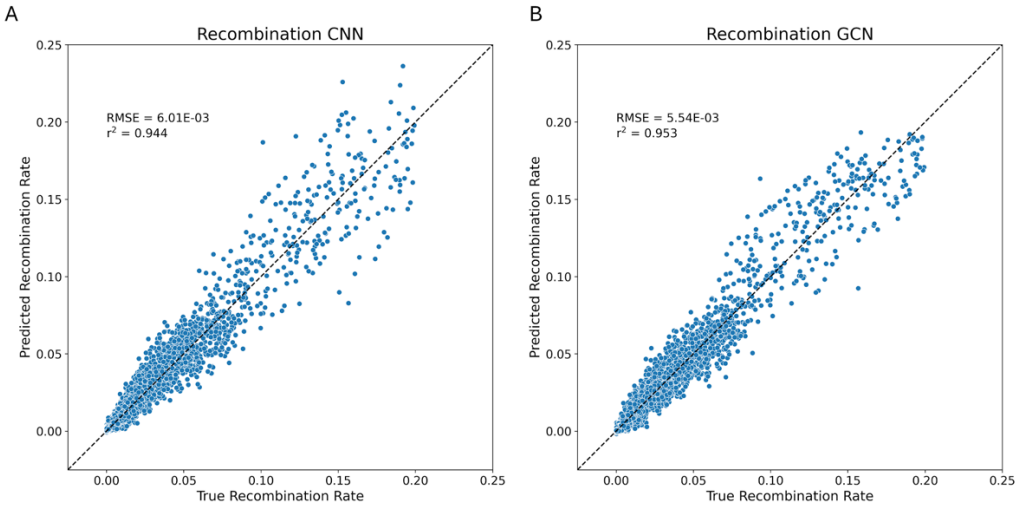


Figure 2. Predicted versus true recombination rates for the (A) CNN and (B) GCN architectures. Predictions are plotted against a reference diagonal dashed line, and each panel shows the root mean squared error (RMSE) and coefficient of determination (R^2) for the given method

To assess the utility of tree-sequence based recombination rate inference, we trained our GCN for this task and compared its performance to the CNN previously shown by Flagel *et al.* (2019) to accurately estimate recombination rates. Specifically, we simulated data using the same parameter values as in Flagel *et al.* (2019)) to generate samples of phased chromosomes which we used to estimate the crossover rate per base pair per meiosis by either passing these data directly into a CNN or through a tree-sequence inference procedure before passing the resulting trees into the GCN. In the context of tree sequences, recombination plays a very direct role in the structure of the data: most recombination events will alter the tree topology, and thus the number of trees in the tree sequence minus 1 is equal to the number of historical recombination events that could possibly be observed. However, because *inferred* tree sequences will contain errors, it is not clear whether we should expect methods that examine information from inferred tree sequences to estimate recombination to be more accurate than methods that directly examine the haplotype data. As shown in Figure 2, our GCN using tree sequences performed slightly better

254 than our CNN (RMSE = 5.54E-03 compared to RMSE = 6.01E-03) that takes the population
255 genetic alignment as input. Thus, the presence of error in inferred tree sequences does not
256 preclude the accurate inference of recombination rates from such data.
257
258 **GCNs dramatically outperform CNNs in detecting selective sweeps**
259 The ability to detect loci affected by positive selection is a longstanding problem in population
260 genetics (Stephan 2019). Selective sweeps occur when positive selection affecting an allele
261 causes it to rapidly increase in frequency in a population, creating a valley of diversity that
262 recovers at increasing genetic distances from the selected site (Smith and Haigh 1974; Kaplan *et*
263 *al.* 1989). Sweeps also produce an excess of high-frequency derived alleles (Fay and Wu 2000),
264 a deficit of haplotypic diversity (Hudson *et al.* 1994; Fay and Wu 2000; Sabeti *et al.* 2002), and
265 elevated linkage disequilibrium (Kelly 1997; Kim and Nielsen 2004) near the target of selection.
266 Recently, there has been great interest in characterizing and detecting the signatures of "soft
267 sweeps" (Hermisson and Pennings 2017) which may be common in large, rapidly evolving
268 populations (Hermisson and Pennings 2005; Karasov *et al.* 2010; Garud *et al.* 2015), and which
269 produce somewhat different signatures than classic "hard" sweeps (Przeworski *et al.* 2005; Berg
270 and Coop 2015). Recently, a number of machine learning methods have been designed to detect
271 the multidimensional signatures of both of these types of sweeps (Pybus *et al.* 2015; Schrider and
272 Kern 2016; Mughal and DeGiorgio 2019; Hejase *et al.* 2022; Whitehouse and Schrider 2022,
273 2023; Caldas *et al.* 2022; Lauterbur *et al.* 2022).
274 However, it should be noted that the presence of all of these signatures of selection is a
275 side effect of the profound effect that positive selection has on genealogies in the genomic
276 vicinity of the selected locus. The shapes and sizes of trees are affected by selection in that the

genealogies of loci near the selected site have a very short branches, especially internal branches, while genealogies at intermediate distances from the sweeping allele may have a subset of long branches as a consequence of recombination during the sweep (Fay and Wu 2000; Przeworski *et al.* 2005; Stephan 2019). Moreover, because of the reduced time period for historical recombination events to occur during the sweep, trees near the selected locus will correspond to a larger stretch of sequence than those from more distant loci that have experienced more recombination (Sabeti *et al.* 2002; Przeworski *et al.* 2005; Ferrer-Admetlla *et al.* 2014; DeGiorgio and Szpiech 2022). Thus, tree sequences may provide a great deal of information about recent positive selection.

To assess the potential of GCNs for detecting sweeps, we used the same simulated benchmarking scenario as Flagel *et al.* (2019), and trained our networks to classify regions as undergoing a recent hard selective sweep (from a *de novo* mutation), being linked to a recent hard sweep, undergoing a soft sweep (from standing variation), being linked to a soft sweep, or evolving neutrally. Specifically, we simulated data under the JPT model from Schrider and Kern (2017) to generate training, testing, and validation sets as described in the Methods. We find that the GCN architecture outperforms the CNN for all classes except neutrality, where their performance is equal (Figure 3). Indeed, the GCN obtains an average increase of 6.4% of test samples assigned to the proper class (the confusion matrix diagonal) relative to the CNN (65.5% accuracy for the GCN vs 58.8% for the CNN). Perhaps more strikingly, the GCN exhibits a sizable increase relative to the CNN in the area under the receiver operating characteristic (ROC) curve (AUROC) and in average precision (AP, equal to the area under the precision-recall curve) for distinguishing sweeps from neutrality (difference in AUROC of 0.085 and difference in AP of 0.151; Figure 4A, B). The increase in accuracy is less notable for distinguishing between hard

and soft sweeps, with an AUROC increase of 0.004 and AP increase of 0.007 for the GCN compared to the CNN (Figure 4C, D).

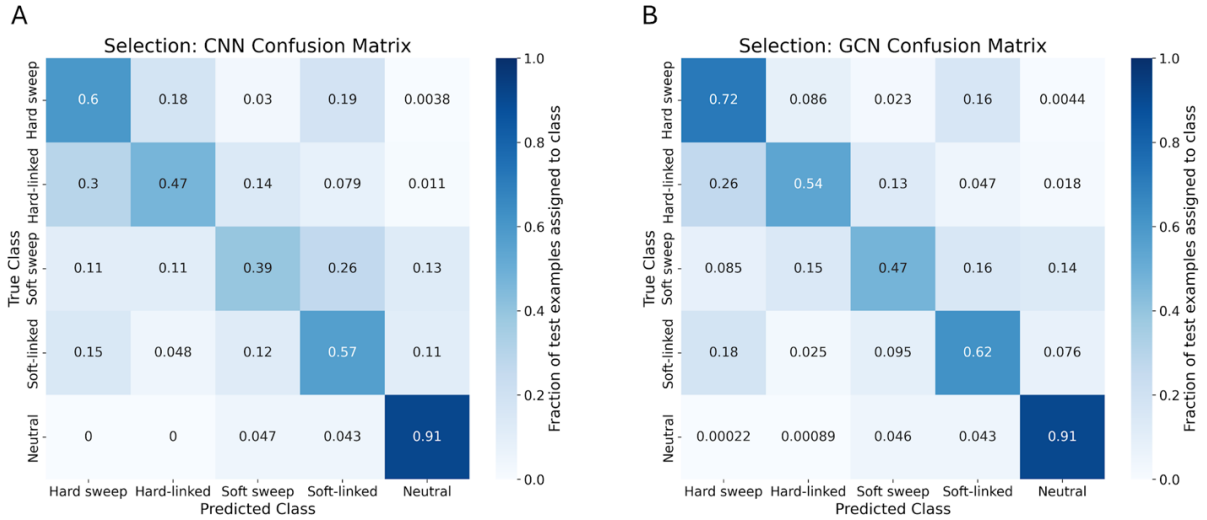


Figure 3. Confusion matrices showing performance of the CNN architecture and GCN architecture on the sweep detection task. A) Confusion matrix summarizing the CNN's performance on a held-out test set. For this problem, there are five classes specifying the type of sweep (hard versus soft), whether a focal window is impacted by direct selection within the window or linked selection outside of the window (sweep vs. linked), or neutrally evolving. Each entry in the matrix shows the fraction of simulated examples under a given class (the True Class, specified by the row of the matrix) that were assigned to a given class by the network (the Predicted Class, specified by the column of the matrix). Diagonal entries thus represent correct classifications, and errors are represented in the off-diagonal entries, and each row sums to 1. B) Confusion matrix summarizing the GCN's performance on the same test set.

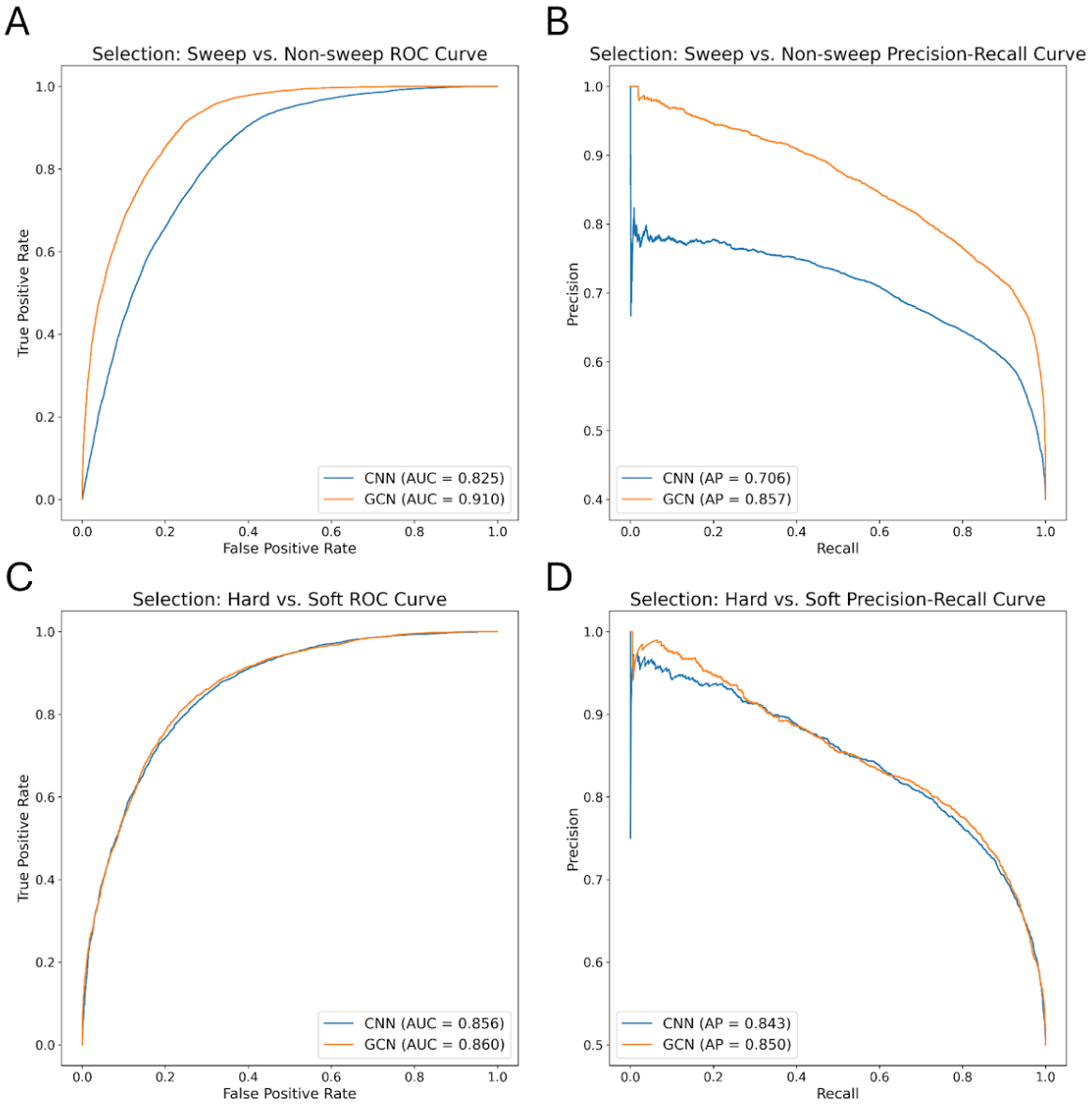


Figure 4. Receiver operating characteristic (ROC) and precision-recall (PR) curves summarizing performance of the CNN (blue) and GCN (orange) on the sweep-detection task. (A) ROC curves showing how well each method distinguishes between selective sweeps (whether hard or soft) and unselected regions (whether sweep-linked or neutral). The curves show the true and false positive rates for the CNN (blue) and GCN (orange) at varying classification thresholds. The score used to classify a window as a sweep vs. unselected at a given threshold was the sum of the neural network's class membership probabilities for the "hard sweep" and "soft sweep" classes to yield a combined "sweep" probability. The area under the ROC curve (AUC) for each network is shown in the inset. B) Precision-recall curves showing the two network's performance on the same "sweep vs. unselected" task, here showing the precision (i.e. the positive predictive value, or fraction of regions classified as sweeps that truly were sweeps) and recall (the true positive rate) at varying classification thresholds. The average

precision (AP, which is equal to the area under the PR curve) is shown in the inset. C) ROC curves for distinguishing between hard and soft sweeps, with hard sweeps treated as positives and soft sweeps as negatives for this binary classification task. D) Precision-recall curves for the "hard vs. soft" task.

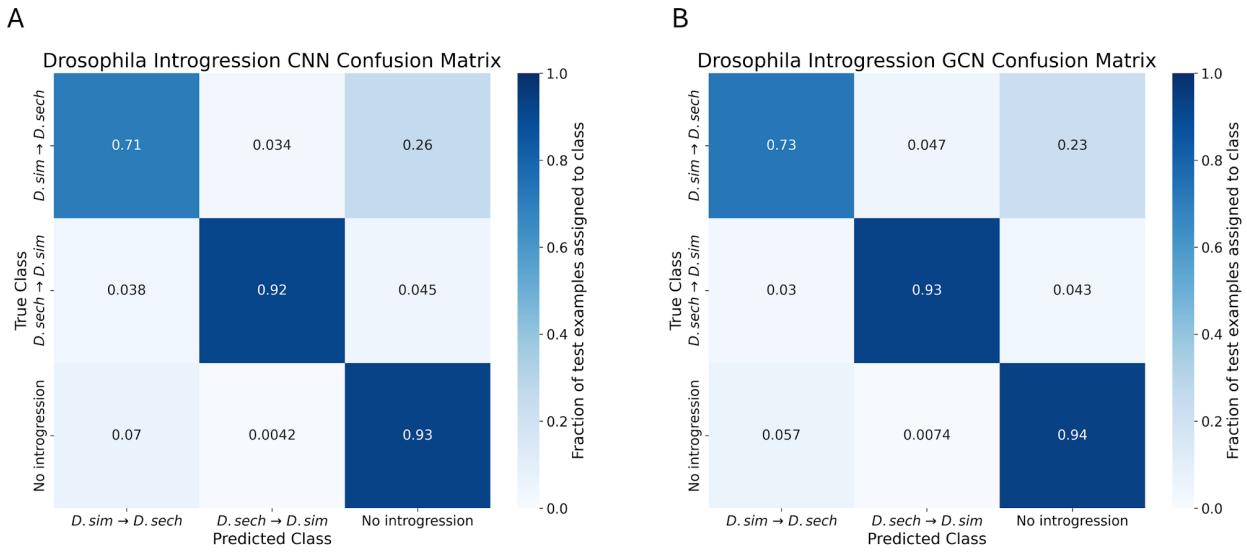
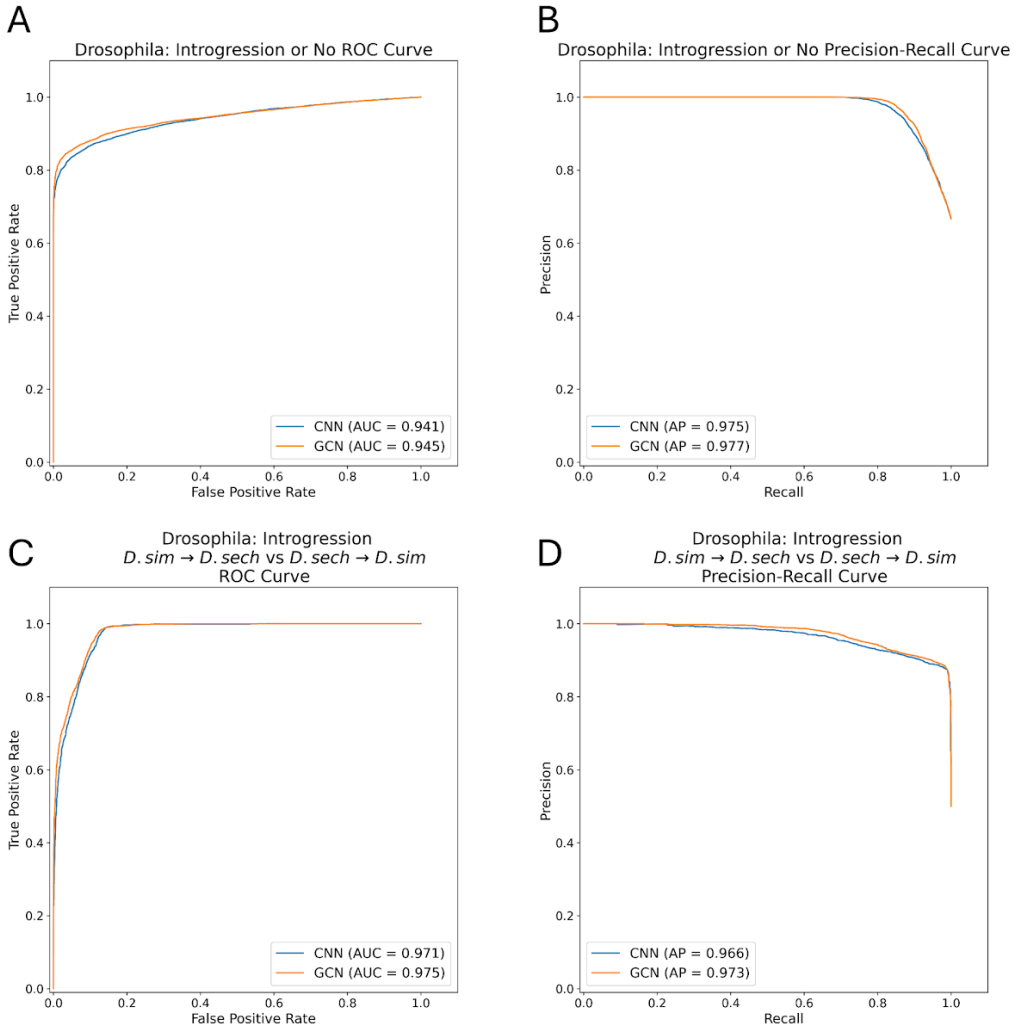


Figure 5. Confusion matrices showing the neural network's accuracies on the introgression inference task. Confusion matrices are shown in the same manner as for Figure 3, but here the three classes are introgression from *Drosophila simulans* to *D. sechellia*, introgression from *D. sechellia* to *D. simulans*, and no introgression. A) Confusion matrix for the CNN. B) Confusion matrix for the GCN.

GCNs perform similarly to CNNs on the task of detecting introgression

A variety of approaches have therefore been developed to identify the signatures of introgression between pairs of closely related species (Hudson *et al.* 1992; Huson *et al.* 2005; Rosenzweig *et al.* 2016), including several machine learning methods (Schrider *et al.* 2018; Gower *et al.* 2021; Ray *et al.* 2023). Previously, Flagel et al. (2019) showed that a CNN substantially outperforms FILET, a supervised machine learning method that detects introgression between closely related

347 species with impressive accuracy (Schrider et al. 2018). Here we compared the performance of



348

349 **Figure 6. Receiver operating characteristic (ROC) and precision-recall (PR) curves for the**
350 **introgression detection task.** A) ROC curves summarizing the performance of the CNN (blue)
351 and GCN (orange) on the task of distinguishing between windows with introgression (in either
352 direction) and without introgression. The area under the ROC curve (AUC) for each network is
353 shown in the inset. B) Precision-recall curves for the task of distinguishing between windows
354 with and without introgression. The average precision (AP, which is equal to the area under the
355 PR curve) is shown in the inset. C) ROC curves for the task of distinguishing between
356 introgression from *D. simulans* to *D. sechellia* and introgression from *D. sechellia* to *D.*
357 *simulans*. D) Precision-recall curves for the task of inferring the direction of introgression.
358

359 the tree-sequence based GCN to our CNN on simulated data modeling gene flow between
360 *Drosophila simulans* and *Drosophila sechellia* (see Methods for details). We find that both
361 networks perform well (Figures 5,6), but as is the case for the other tasks the GCN outperforms
362 the CNN (86.2% accuracy for the GCN versus 85.1% for the CNN). The overall error profile is
363 very similar for the two architectures (Figure 5) indicating that the GCN's improvement over the
364 CNN is not due to any specific class, but rather due to slight accuracy increases (1–2%) across
365 all three classes. The GCN has an AUROC increase of 0.004 and AP increase of 0.002 when
366 measuring accuracy distinguishing between genomic windows with and without introgression,
367 and an increase in AUROC of 0.004 and AP of 0.007 when inferring the direction of
368 introgression (Figure 6).
369
370 **GCNs show improvement compared to CNNs on a demographic inference task**
371 Perhaps one of the most widely studied tasks in population genetics is the inference of
372 demographic models and their associated parameters from. Many statistical approaches have
373 been developed to allow researchers to estimate a population's history of size changes, splitting
374 and merging events, and expansions and contractions (Gutenkunst *et al.* 2009, 2010; Li and
375 Durbin 2011; Browning and Browning 2015; Terhorst *et al.* 2017; Kamm *et al.* 2020; Santiago *et*
376 *al.* 2020; Excoffier *et al.* 2021) These include several deep learning methods that have been
377 designed in recent years (Sanchez *et al.* 2021; Wang *et al.* 2021), one of which applies graph
378 convolutions to the tree sequence for inference (Korfmann *et al.* 2023b). To compare the
379 performance of our GCN to the CNN on a demographic inference task, we reexamined the 3-
380 epoch model presented in Flagel *et al.* (2019). We simulated data under this model and assessed

each network's accuracy in estimating its 5 parameters: the population sizes in each of the 3 epochs, and the 2 transition times between these epochs.

We found that the GCN architecture outperforms the CNN on all estimations except the ancestral population size (N_2) where the two methods perform equally as measured by both root mean squared error (RMSE) and the coefficient of determination (R^2) on test data (Figure 7). With respect to population size inference, the difference in accuracy between the GCN and CNN was most pronounced for the present-day population size (N_0; improvement in RMSE and R^2 of 0.046 and 0.048, respectively, for the GCN over the CNN), with the middle epoch's population size inference (N_1) showing smaller differences in accuracy (difference in RMSE and R^2 of 0.026 and 0.045, respectively), and again no difference between the two methods for the ancestral size. For the inferred times of population size changes, we again observed greater difference in accuracy for the inferred time of the most recent event (T_1; difference in RMSE and R^2 of 0.067 and 0.069, respectively) than for the inferred time of the more ancient size change (T_2; difference in RMSE and R^2 of 0.043 and 0.047). This improvement in accuracy is somewhat modest (average RMSE decrease of 0.055 and average R^2 decrease of 0.058 across all parameters), and we remind the reader that this test case involves the inference of demographic parameters from a single genomic region as a proof of concept rather than examining genome-wide data as is typically done in practice. Nonetheless, these results demonstrate the GCN's ability to examine inferred tree sequences and extract information relevant to past demographic events at least as well as a CNN directly examining the population genetic alignment.

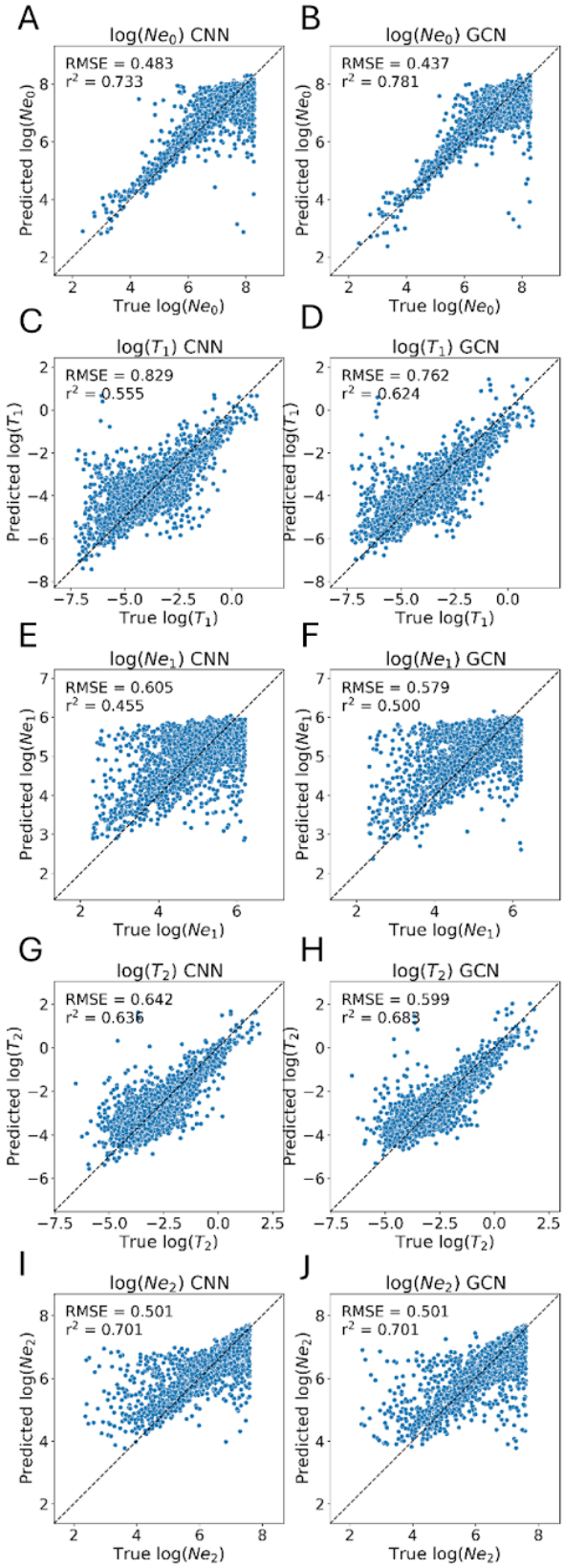


Figure 7. Performance of the CNN and GCN on the task of demographic parameter estimation on a held-out test set. The top row shows the predicted versus true values for the CNN (panel A) and GCN (panel B) for the present-day population size N_0, and subsequent rows show the same for the time of the most recent population size change (T_1, panels C and D), the second population size (N_1, panels E and F), the time of the more ancient population size change (T_2, panels G and H), and the ancestral population size (N_2, panels C and D). Each panel also shows the root mean squared error (RMSE) and coefficient of determination (R^2) for the given method on the given parameter. Values are log-scaled for visual clarity and plotted against a reference diagonal dashed line.

Performance and computational efficiency

Perhaps the most enticing attribute of tree sequences is that they can provide extremely compact representations of population genomic data (Kelleher *et al.* 2019), leading to the possibility that they can be leveraged to reduce the computational burden of evolutionary inference pipelines. We therefore compared the computational cost of our GCN and CNN on the problems examined above. With respect to

437 preprocessing, our GCN's pipeline has an advantage over our CNN pipeline in that tree
438 sequences are relatively fast to infer compared to the sorting and matching steps that we perform
439 on alignments before passing them into the CNN. For instance, computing Relate outputs for
440 1000 simulated replicates of length 10 kb from the introgression case took on average ~ 5
441 minutes for a single CPU core vs. ~ 21 minutes to seriate and match the same number replicates.
442 In terms of memory efficiency, the size of a single replicate of from the introgression data set
443 after the tree-sequence inference step was 3.25 times the size of the original simulations (Tables
444 2, 3). Similar trends are observed for the recombination and demography problems. For sweep
445 detection, the GCN requires slightly less memory than the CNN, but this is a consequence of our
446 downsampling of the tree sequence for the GCN (Methods), while the CNN examines the entire
447 input alignment.
448 Training time is also an important consideration when choosing deep learning
449 architecture. We note that our CNN, due to the smaller input sizes and the greater efficiency of
450 image convolutions over graph convolutions with current hardware and software, are faster to
451 train and usually require less RAM (Tables 1, 2). This is in spite of the fact that our choice of
452 CNN has almost 2 orders of magnitude more learnable parameters than our proposed GCN
453 architecture (21,280,965 vs. 228,837 respectively). However, we find both approaches to be
454 accomplishable on consumer-grade hardware, i.e. on a single GPU with ~ 12 Gb of RAM.
455

Table 1: GPU training and approximate RAM usage for the GCN architecture. All training times are reported for a single NVIDIA A100 GPU. RAM estimates were obtained via *nvidia-smi* while training was running. For the GCN, batch size in terms of number of input trees varies stochastically due to the variable tree sequence length.

Problem	Time	RAM (1 batch)	RAM (1 replicate)	Batch size	Total epochs
Recombination rate	5.5 hrs	8.7 Gb	145 Mb	60	28
Introgression	12.4 hrs	8.3 Gb	231 Mb	36	43
Sweeps	24.6 hrs	10.7 Gb	535 Mb	20	67
Demography	14.5 hrs	8.4 Gb	280 Mb	30	100

Table 2: GPU training and approximate RAM usage for the CNN architecture. All training times are reported for a single NVIDIA A100 GPU.

Problem	Time	RAM (1 batch)	RAM (1 replicate)	Batch size	Total epochs
Recombination rate	0.86 hrs	3.07 Gb	51 Mb	60	18
Introgression	0.66 hrs	3.4 Gb	71 Mb	48	13
Sweeps	7.1 hrs	24.2 Gb	605 Mb	40	18
Demography	0.85 hrs	3.5 Gb	109 Mb	32	23

DISCUSSION

The size and quantity of population genetic datasets are increasingly large, making downstream analysis more computationally intensive. As researchers develop new, more compact data structures for population genetics, new methods must be developed to fully take advantage of these structures' potential to tackle long standing problems in the field with greater computational efficiency and perhaps better accuracy as well. Tree sequences have become

471 increasingly popular, both in the context of estimation from empirical data and in simulation
472 (Kelleher *et al.* 2018; Haller *et al.* 2019; Ralph *et al.* 2020). However, there has yet to be a
473 fundamental shift in population genetics inference approaches away from less efficient data
474 structures (Korfmann *et al.* 2023a). In principle, tree sequences have a key advantage over
475 genotype information in that they can represent the complete evolutionary history of a sample of
476 genome sequences, making them an attractive avenue for downstream evolutionary inference.
477 Several previous studies have presented deep learning approaches for analyzing tree-
478 sequence data, with initial methods using predefined sets of features summarizing the properties
479 of trees in the sequence that are then fed into a neural network (Hejase *et al.* 2022; Mo and
480 Siepel 2023; Pearson and Durbin 2023). While these approaches showed great promise, because
481 they examine summaries of the trees rather than the tree itself, they could potentially omit
482 valuable information present in the tree sequence. Just as deep learning-based methods operating
483 directly on genotype/haplotype matrices are able to extract more useful information than vectors
484 of statistics designed to summarize aspects of a sample of genotypes/haplotypes (e.g. allele
485 frequencies, linkage disequilibrium) (Kern and Schrider 2018; Flagel *et al.* 2019), graph neural
486 networks operating directly on tree sequences may perform better than methods using predefined
487 tree summaries (e.g. the number of lineages present in the tree within each of a set of non-
488 overlapping time intervals (Hejase *et al.* 2022)). Indeed, this approach was used successfully by
489 (Korfmann *et al.* 2023b). for demographic inference. Here, we adopt a similar strategy, using a
490 GCN to show that tree sequence-based inference appears to be at least as powerful of a
491 framework for evolutionary inference as deep neural networks acting directly on a population
492 genetic alignment.

493 The main caveat with using tree sequences is that in empirical data the true evolutionary
494 history of a sample can never be known, and therefore tree sequences must be estimated from
495 genotypes (Kelleher *et al.* 2019; Mahmoudi *et al.* 2022). As it stands, inference methods can
496 produce highly inaccurate tree sequences when benchmarked on simulated data (Zhang *et al.*
497 2023), suggesting that using tree sequences as input for further inferences may be problematic.
498 Moreover, the paradigm of inferring a population sample's tree sequence to use as input for a
499 downstream inference task does on the face of it appear to be unnecessarily complicated, and
500 appears to violate the following imperative from Vapnik: "When solving a problem of interest,
501 do not solve a more general problem as an intermediate step" (Vapnik 2006). For example, if we
502 wish to infer whether a given locus has experienced a selective sweep, it may not be desirable to
503 first infer the full evolutionary history of every segment of DNA along a large stretch of
504 chromosome surrounding this locus, as this would clearly be a more challenging task than
505 directly searching for evidence of recent positive selection.
506 Why then, might tree sequence-based inference still be generally highly effective, as our
507 results seem to suggest? First, it seems probable that incorrect tree sequences contain information
508 that, although incorrectly describing the true evolutionary history of a sample, can be useful for
509 population genetic inference. In addition, tree sequences allow for a more concise representation
510 of population genetic data than a genotype matrix. Moreover, tree sequence inference may serve
511 as a form of de-noising by aggregating data across sites in a region in order to extract a
512 meaningful pattern that may not be easily observable directly from sequence data (especially in
513 the presence of sequencing error), even if that pattern does not describe the true evolutionary
514 history. In any case, supervised machine learning methods combined with tree sequences can be
515 especially powerful because one can train the machine learning model on tree sequences that are

516 inferred from simulated data (as we have done here), rather than the true tree sequences, so that
517 the training tree sequences may exhibit a similar error profile as tree sequences inferred from real
518 data. In this framework, it does not matter whether the tree sequences are correct, but only that
519 they contain information useful for the task at hand. Moreover, the impact of any remaining
520 fundamental differences between tree sequences from simulated vs. real data could potentially be
521 mitigated via the machine learning technique of domain adaptation (Ganin and Lempitsky 2015),
522 wherein a network can be trained to learn to ignore systematic differences between simulated and
523 real data before downstream prediction (Mo and Siepel 2023). Thus, in spite of the somewhat
524 unusual workflow, relying on inferred tree sequences as input to supervised machine learning
525 methods may in fact prove to be an especially powerful and robust strategy.

526 We note that although we have provided an attempt at solving multiple population
527 genetics inference tasks the number supervised machine learning approaches to population
528 genetics has ballooned in recent years such as detecting selective sweeps (Kern and Schrider
529 2018; Hejase *et al.* 2022; Whitehouse and Schrider 2023), detecting introgression (Schrider *et al.*
530 2018; Gower *et al.* 2021; Ray *et al.* 2023), predicting locations of recombination (Adrion *et al.*
531 2020) among others (reviewed in Schrider and Kern 2018; Flagel *et al.* 2019; Korfmann *et al.*
532 2023a). The encouraging benchmarking results from our GCN suggest that machine learning
533 methods using tree sequences will likely prove to be a powerful addition to this ever-growing
534 suite of tools and tasks.

535 One important limitation of our work is that, although our networks were able to be
536 trained and executed on consumer-grade hardware, we have not made exhaustive attempts to
537 improve computational efficiency. Indeed, we find that our CNN is more efficient than the GCN
538 in terms of both memory usage and training time, with the current advantage of the GCN being

its faster preprocessing time. However, the latter is a consequence of our choice to sort our alignments in order to improve performance, a computationally expensive process (Ray *et al.* 2023) and it is important to note that the cost of this step will depend on the sorting algorithm used and the size and number of alignments to be sorted. Moreover, at least some problems, exchangeable neural networks that do not require any sorting may be used (Chan *et al.* 2018). However, we note that the advantage in compactness afforded by tree sequences is not observed at small sample sizes (Kelleher *et al.* 2019), and the tasks examined here all used small to modest sample sizes. Thus, our conclusions about the relative efficiency of GCNs vs. CNNs may not hold for tasks requiring much larger samples. Further investigations into optimizations of both architecture and processing algorithms may increase the favorability of the GCN in all aspects. future efforts experimenting with more compact tree representations (Kelleher *et al.* 2018, 2019; Haller *et al.* 2019; Ralph *et al.* 2020; Mahmoudi *et al.* 2022) and different neural network architectures may yield improvements in this area.

Finally, we note that we have only explored a relatively small number of population genetic inference tasks that are of particularly broad interest to researchers. We therefore encourage exploration of tree sequence-based deep learning inference as a general framework for population genetic inference. Our hope is that this may not only lead to improved accuracy on the problems examined here, but that tree sequence-deep learning paradigm may prove useful for a wider array of problems and, as has proved to be the case for alignment-based deep learning applications, even empower researchers to tackle new problems (Battey *et al.* 2021; Yelmen *et al.* 2021; Korfmann *et al.* 2023a; Smith *et al.* 2023; Booker *et al.* 2023; Whitehouse and Schrider 2023).

METHODS

Tree sequence inference

For each problem, simulations were performed as described in the sections below, producing *ms*-style output. The simulation output was then formatted for input to *RELATE* (Speidel *et al.* 2019), a recently developed method that, when given a sample of genomes along with estimate of the mutation rate, recombination rate, and effective population size, infers the sample's genome-wide genealogy in the form of a sequence of rooted binary trees. *RELATE* outputs this sequence of trees in order along the chromosome, with each tree spanning a variable length of sequence bounded by the breakpoints of inferred historical recombination events that altered the tree topology. This was done for each simulation replicate for each problem described below, using the mutation, recombination rate and effective population size parameters listed in Table S1.

Overview of Graph Convolutional Networks (GCNs)

Graph convolution (Kipf and Welling 2017) works by aggregating information at each node in a graph by combining data from its neighbors (those nodes with which it shares an edge). The input to a graph convolution operator is a graph or a set of nodes and edges (*N*, *E*), where each $x \in N$ is some n-dimensional vector containing information about a given node, and each element of *E* is a pair of node indices representing a connection between those two nodes. The edges of a graph are often specified instead by an adjacency matrix $\mathbf{A}$ which specifies the topology of the graph: i.e. $\mathbf{A}$ has a value of 1 at entry ij if node i and node j share an edge and a value of 0 otherwise. Formally the graph convolution can be written in matrix form as:

$$\mathbf{X}' = \hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\mathbf{X}\mathbf{\Theta} \quad (1)$$

where $\mathbf{X}$ is a matrix consisting of the n node feature vectors $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ with I as the identity matrix and $\hat{D}_{ii} = \sum_{j=0} \hat{A}_{ij}$ as the diagonal degree matrix. Here features are first aggregated according to the connectivity specified in $\mathbf{A}$ including the features belonging to the node itself (hence the addition of the identity matrix specifying self-connectivity, a "self-loop") and then normalized using the degree matrix to avoid exploding gradients. Then, they are transformed according to some learnable matrix $\mathbf{\Theta}$ that is optimized through gradient descent during training. For clarity, the node-wise formula is:

$$\mathbf{x}'_i = \mathbf{\Theta}^\top \sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{1}{\sqrt{\hat{d}_j \hat{d}_i}} \mathbf{x}_j \quad (2)$$

where the degree of node (including self-connectivity) i is $\hat{d}_i = 1 + |\mathcal{N}(i)|$, with $\mathcal{N}(i)$ being the set of neighbors of node i, and x_i being the node features for node i. Thus, in each graph convolution step, each node's features are updated to the weighted sum of all its neighbors features including its own (i.e. the features of the set of nodes *, as specified in the summation)., Specifically, where each contribution of each node j's feature vector is weighted by the reciprocal of the square root of the product of node *'sits degree and node i's degree and the degree of its neighbor, as in the denominator of Equation 2. Then, that resulting sum is transformed according to the learnable matrix $\mathbf{\Theta}$. This is called a "convolution" on graphs as it is analogous to convolutions on images where the "kernel" of a graph convolution operates over each local neighborhood of each node much like an image convolution kernel operates on some rectangular region around each pixel (see example in Figure S1).

In the case of genealogical trees, the node features returned by each graph convolution operation are a function of those of its parent and its two children if it is an internal node, its two children only if it is the root node, and its parent only if it is a leaf node. For our input graphs we

used the following features: the age of a node (the time in generations that the coalescence event associated with it occurred, or 0 for leaf nodes), the number of mutations on the branch from its parent (mean and standard deviation normalized via the training data), and a one-hot encoded label vector specifying whether the node is internal (i.e. ancestral) or external (present day) respectively, and the population label if the node is external (for internal nodes the population is considered unknown). Thus, if there are k populations the size of the label vector is $k + 1$. Note that the definition of graph convolution allows for directionality of the edges, i.e. one can specify whether node i passes information to node j in each graph convolution operation or vice-versa. Here we specify that the edges are unidirectional with the direction being child to parent. Thus, for our GCN $\mathcal{N}(i)$ is the set of child nodes of node i., Fand no update is performed for external nodes in the tree $\mathcal{N}(i)$ is the empty set, and thus for those nodes $\mathcal{N}(i) \cup \{i\} = \{i\}$, meaning only the focal node is considered during the convolution.

GCN Network Architecture and training

The GCN architecture implemented here (illustrated in Fig. 1) consists of two components: a graph-learning network, or layers that operate on graphs, and two recurrent networks that operate on sequences of feature vectors. First, each tree in the input sequence has its node features transformed via a learned affine transform with bias, to which the original features are concatenated to via a "skip connection" before each tree is then input separately to a series of independent graph convolution layers with skip connections, comprising a graph convolution "block" of layers. At each layer, the output from the previous layer is transformed via convolution and added to itself, then the result is normalized and passed to the ReLU activation function (Agarap 2018). For normalization in the GCN layers we use LayerNorm (Ba *et al.*

2016). The graph convolution layer we chose for our architecture is the Graph Attention Convolution (GATConv) from (Brody *et al.* 2022) implemented in Pytorch (Paszke *et al.* 2019). It is a modified form of the proposed convolution operator that we detail in the previous section in that node features from each neighbor are linearly transformed *before* they are summed in the update step and are weighted by learned attention coefficients instead of simply the degree of the nodes, enabling the network to potentially learn which neighbor's features are more or less relevant for accomplishing the given regression task. In short, weighting by node degrees as described in the previous section prevents gradient explosion whereas weighting by attention prevents gradient explosion *and* has the benefit of learning the most optimally-informative connections in the graph for a given task (Brody *et al.* 2022). Formally the convolution with attention is specified as:

$$\mathbf{x}_i' = \alpha_{i,i}\mathbf{\Theta}_s\mathbf{x}_i + \sum_{j\in\mathcal{N}(i)} \alpha_{i,j}\mathbf{\Theta}_t\mathbf{x}_j \quad (3)$$

Where the attention coefficients α are computed as:

$$\alpha_{i,j} = \frac{\exp\left(\mathbf{a}^\top \mathrm{LeakyReLU}\left(\mathbf{\Theta}_s\mathbf{x}_i + \mathbf{\Theta}_t\mathbf{x}_j\right)\right)}{\sum_{k\in\mathcal{N}(i)\cup\{i\}} \exp\left(\mathbf{a}^\top \mathrm{LeakyReLU}\left(\mathbf{\Theta}_s\mathbf{x}_i + \mathbf{\Theta}_t\mathbf{x}_k\right)\right)} \quad (4).$$

With $\mathbf{a} \in \mathbb{R}^d$ and $\mathbf{\Theta}s, \mathbf{\Theta}_t \in \mathbb{R}^{d'\times d}$ being learnable parameters weighted during training where d and d' are the dimension of the input features and output features respectively and R^d being a d-dimensional vector of real numbers. In each GATConv layer, attention coefficients $\alpha_{i,j}$ are computed which selectively filter messages from neighboring nodes. This attention mechanism is comprised of the same two learnable affine transformation matrices used in eq. 5, $\mathbf{\Theta}_s$ and $\mathbf{\Theta}_t$, which transform the focal node and its neighbor's node features respectively and **a** being a trainable weight vector onto which the summed features are projected via the dot product and

then normalized over all neighbors to sum to one using the softmax function. In this way, the attention mechanism hopes to weight the messages from neighboring nodes with $\alpha_{i,j}$ such that returned features have the most impact on loss reduction for a given target. Note that self-attention coefficients expressing how much the focal node's own features contribute to the features in the next layer ($\alpha_{i,i}$) are also computed in a similar manner.

After passing through the GATConv layers, features obtained from the nodes in each tree are separately passed through a Gated Recurrent Unit (GRU; (Cho *et al.* 2014)) and the hidden features are returned as the compressed vector representation of each tree in the sequence. For each GRU's input, the order of nodes in the tree is the same order that is returned by RELATE in Newick format. The vector emitted by each GRU is concatenated to a "tree summary vector" containing the following features about the original tree: the time of coalescence of all individuals in the tree, the mean, median, and standard deviation of coalescence times, the mean, median, standard deviation, skew, and maximum of the branch lengths in the tree, the midpoint of the chromosomal segment the tree corresponds to (scaled from 0 to 1, with 0 being the left end of the simulated chromosome and 1 being the right end), and the fraction of the entire sequence accounted for by this segment. The resulting vectors, one for each input tree, are then concatenated into a single matrix, which is in turn passed through another GRU that emits a feature vector summarizing the entire tree sequence. A global vector (the "tree sequence summary vector") which consists of the mean, standard deviation, and median of the tree summary vectors, as well as the number of graphs in the original sequence, is embedded using an affine transform with bias and then concatenated to the resulting hidden vector from the previous GRU, resulting in a 1D vector containing learned information from the graph convolution process and summary statistics about the tree sequence as a whole. Each global vector and pre-

computed tree summary vector for the graphs is mean and standard deviation normalized via the training data set in preprocessing. Finally, this is passed through three fully connected layers (a multi-layer perceptron, "MLP") with the first two using ReLU activations and batch normalization (Ioffe and Szegedy 2015), and the final (output) layer using no activation function. This results in classification scores which could be interpreted as posterior probabilities by applying the softmax function, or in the case of regression, *z*-scores for the various values to be predicted (all y values for the regression tasks are standardized via the mean and standard deviation computed from the training set). We provide a table specifying the dimensionality of the inputs and outputs in each layer of our network as well as the activations used for clarity (Table 1).

When passing the ragged sequences to the final GRU layer they are each padded to the maximum length observed in the given batch. For all the experiments however, we first downsample the tree sequences if they are > 128 trees in length by randomly selecting a sequential window of 128 trees and omitting the others. This cap on tree sequence lengths allowed us to increase the batch size while still being able to train using a single GPU. For some tasks, namely in the case of recombination rate and demographic parameter regression, this is far more than the average number of trees observed in the chosen window size and thus most tree sequences are passed in full to the network. However, in the case of the sweep detection problem, there are often far more than 128 trees in the tree sequence, and thus many of the replicates passed to the network are downsampled sequences. Further details of how we downsample the tree sequences and the average amount of downsampling are given in the sections detailing each task.

For classification tasks, we used categorical cross entropy as our loss function to be minimized during training:

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{M} y_{i,c}\log y_{i,c} \quad (5)$$

Where $y_{i,c}$ is a binary indicator of the correct class and $y_{i,c}$ is our posterior probability estimate that the observation came from this class where i is the replicate index, c is the class index, M is the number of classes, and N is the batch size or number of replicates considered in each gradient step. For regression tasks, we use a variant of L1 loss (i.e. mean absolute error (MAE)) because of L1's relative insensitivity to outliers; specifically, we used the "smooth L1" loss function, whose slope decays at values closer to zero (Girshick 2015). However, note that for all methods and tasks we report final regression accuracies in root mean-squared error (RMSE) to better compare to Flagel et al. (2019). y values for the regression tasks are standardized via the mean and standard deviation computed from the training set. We trained using the ADAM optimizer (Kingma and Ba 2017) with a learning rate of 1e-4 for up to 100 epochs or until their validation accuracy stopped increasing for 10 consecutive epochs. For each problem we specify the batch size in the sections below.

Table 3: Graph neural network layers and their input/output shapes and activation functions.

B = number of tree sequences per batch (batch size)
T = number of trees in a single sequence
T' = maximum number of trees in a sequence in a given batch, used for padding
N = number of nodes in a single tree

$T \times N$ = matrix of trees × nodes comprising all nodes in a single tree sequence (i.e. the number of nodes across all trees in a single sequence)

F = number of node features calculated for each node in the input tree sequence

E = number of edges in all graphs included in the batch of tree sequences

M = the size of the output (the number of classes for classification, or 1 for regression)

(..., n) = number of features

E.g. A shape of (B, $T \times N$, 4) is equivalent to a 3D matrix consisting of a batch (B) of 4 features

layer	type	input shape(s)	output shape	activation
node embedding	Linear	(B, $T \times N$, F)	(B, $T \times N$, 26)	none
skip	concatenation	(B, $T \times N$, F), (B, $T \times N$, 26)	(B, $T \times N$, 26+F)	none
graph convolutions (x6)	GATv2Conv (Veličković *et al.* 2018)	(2, E), (B, $T \times N$, 26+F)	(B, $T \times N$, 26+F)	ReLU
skip	concatenation	(B, $T \times N$, F), (B, $T \times N$, 26+F)	(B, $T \times N$, 26+2F)	none
reshape		(B, $T \times N$, 26+2F)	(B, T, N, 26+2F)	none
RNN	GRU (1-layer, unidirectional)	(B, T, N, 26+2F)	(B, T, 256)	none
tree-level statistics	concatenation	(B, T, 12), (B, T, 256)	(B, T, 268)	none
reshape and pad sequences		(B, T, 268)	(B, T', 268)	none
RNN	GRU (1-layer, unidirectional)	(B, T', 268)	(B, 256)	none
tree-sequence-level (global) statistics	concatenation	(B, 37), (B, 256)	(B, 293)	none
MLP (x2)	Fully-Connected	(B, 293)	(B, 256)	ReLU

output	Fully-Connected	(B, 256)	(B, M)	none

Because input tree sequences are of variable length, here we define the dimensionality of a single batch as (B, T×N) where B is the batch size (number of tree sequences in a batch) and T×N is the number of trees × the number of nodes. For each node, we calculate a set of F features, giving us an input shape of (B, T×N, F) as shown in the "input shape" column for the first layer Table 3 (the "node embedding" row). The inputs to the graph convolution layers are a 2-D tensor of shape (T×N, 26+F), where T×N is again the total number of nodes in the tree sequence, and 26 features per node resulting from concatenated with the original node features from the first skip connection), and a list of edge indices of shape (2, E) specifying the pairwise connections between nodes where each row is a pair of node indices the edge connects, similar to an adjacency matrix this information is used by the graph convolution layers to utilize the node connectedness. The input and output shapes of the skip connection layer, reshaping layer, and first GRU layer that follow the graph convolution layers are also determined by T, N, and F as shown in Table 1. When considering the input to the second GRU layer, T' refers to the padded tree sequence length, which is the maximum observed number of trees among the B tree sequences in a given batch. This padding is performed because all tree sequences in a given batch must have the same length, and we chose to increase the length of tree sequences to T' by concatenating tensors filled with zeros to the end of the GRU's input tensor rather than downsampling the tree sequence by removing trees. Note that this padding was done using PyTorch's `pack_padded_sequence` function that allows for more efficient computation by allowing T' to vary across batches rather than setting it to the maximum length across all batches and instructing the GRUs to ignore the padded part of the sequence during forward and backward computations (Paszke *et al.* 2019). This is followed by another concatenation layer and

then several fully connected layers culminating in the output layer. The input and output shapes for all of these are shown in Table 1. All benchmark tasks except the introgression-detection task use identical output shapes for all layers except the final output layer, which is task-dependent. In total, the chosen architecture has 228,837 parameters for the demography and positive selection problems, 229,771 parameters for the introgression classification problem, and 228,381 parameters for the recombination rate problem.

CNN training

The CNN used for comparison was the Pytorch (Paszke *et al.* 2019) implementation of the ResNet34 architecture (He *et al.* 2015). ResNets are a type of CNN that feature skip or residual connections that merge output from the convolutions with the output from previous layers via addition, and have been shown to perform competitively on traditional image recognition tasks (Reddi *et al.* 2020). The model architecture is illustrated in Figure S2. Models were trained for up to 100 epochs or until their validation accuracy stopped increasing for 10 consecutive epochs. For classification tasks (selection detection, introgression detection) categorical cross entropy loss was used, while smooth L1 (Girshick 2015) loss was used for regression tasks (rho estimation, demographic inference). The ADAM optimizer with default settings (lr=0.001, betas=(0.9, 0.999)) (Kingma and Ba 2017) was used for all networks. All genotype matrix inputs were formatted to be of shape (populations, individuals, polymorphisms) and were padded with zeros to the max observed number of polymorphisms in each case. The rows were sorted based on cosine distance using seriation as described in (Ray *et al.* 2023) using Googles OR-tools package (Perron and Furnon 2019); note that the sorting algorithm and similarity metric differ from that used by (Flagel *et al.* 2019) as we found the seriation approach to perform better on the

task of identifying introgressed haplotypes (Ray *et al.* 2023). Data was represented as binary encoding of 0 or 1 (ancestral or derived) for all tasks.

The batch size for the CNN for both regression cases was 32. The batch size for the selection problem was 10 as the padded size was very large at 5000 SNPs. The batch size for the introgression problem was 96. The CNN used for comparison was the ResNet34 architecture (He *et al.* 2015) which was implemented in PyTorch. ResNets are a type of CNN that feature skip or residual connections that merge output from the convolutions with the output from previous layers via addition, and have been shown to perform competitively on traditional image recognition tasks.

Test Cases

Inferring the historical recombination rate

Simulations for training and benchmarking performance on recombination rate inference were done using *ms* (Hudson 2002) with the same simulation commands used by Flagel *et al.* (2019); these commands are available at https://raw.githubusercontent.com/flag0010/pop_gen_cnn/master/historical_recombination/phased/run.ms.sh. In particular, for each simulation replicate a population size N was drawn from the set {5000, 10,000, 20,000, 50,000}, and, after setting the locus size to L=50 kb and the mutation rate to $\mu=1.5\times10^{-8}$, a recombination rate $\rho=4NrL$ was chosen such that the per-base pair crossover rate, r, was drawn from a truncated exponential distribution with lower and upper boundaries of 10^{-8} and 10^{-6}, respectively. The sample size was set to 50 haploid chromosomes. Simulation and formatting resulted in 189,528 replicates for training the CNN, and 9,956 for its validation.

782 Recombination rates values were log-scaled and then standardized to a *z*-score using the
783 training set mean and standard deviation. Note that the validation and test data were also
784 standardized using the training set's mean and standard deviation to prevent data leakage from
785 the training to the validation/test data. Because the number of segregating sites differed across
786 simulation replicates, the data was padded to 376 polymorphisms or the maximum observed
787 number of polymorphisms in the simulated replicates. The median number of trees returned by
788 RELATE (computed from training data) was 13 with a standard deviation of 24.05. For the
789 GCN, the training and validation sets had 174,575 and 18,625 replicates respectively. The test
790 set used to compare the two approaches had 9,715 replicates.
791
792 *Detecting selective sweeps*
793 We benchmarked accuracy on the task of detecting selective sweeps by adopting the approach of
794 (Schrider and Kern 2016; Flagel *et al.* 2019) in that we define the problem as a classification task
795 with 5 classes: hard sweeps (a sweep of a *de novo* mutation), hard-linked regions (i.e. those close
796 to a hard sweep), soft sweeps (selection on standing variation), soft-linked regions (close to a
797 soft sweep), and neutrally evolving regions. Sweeps were simulated using *discoal* (Kern and
798 Schrider 2016) with the same simulation parameters used by Schrider and Kern (2017) to
799 generate training and test data for the JPT sample (Japanese individuals from Tokyo) from Phase
800 3 of the 1000 Genomes dataset (Auton *et al.* 2015).
801 For the hard and soft sweep classes, the target of selection was located at the center of the
802 simulated chromosome. For the sweep-linked classes the target of selection was located outside
803 of the central "sub-window" (which accounts for 1/11 of the total length of the selected
804 chromosome). For this task we consider a simulated example to be a sweep if the central

polymorphism in the sampled window is the target of selection. Sweep-linked classes are defined as having a sweeping allele in the sampled window but not in the center, while the neutral class does not have any selection in the simulation.

For sweep scenarios we condition on the sweeping allele having reached at time $\tau \sim U(0, 2000)$ generations ago. The sample size was set to 104 haploid individuals. Genotype matrices were either padded or cropped to 5000 polymorphisms. The training, validation, and test sets for the CNN had 236,232, 26,432, and 22,480 replicates respectively.

For the tree sequences, sequences containing more than 128 trees were downsampled as in the other problems, but instead of sequential window sampling we sampled trees by setting the probability that a tree was sampled by the fraction of the chromosome that it spanned, and then sampling with replacement while keeping the original ordering intact. This was done because a large proportion of simulations had >> 128 trees in sequence (the median length across the five categories was 107 with standard deviation of 111.2) and non-linked replicates can have selection almost anywhere along the simulated chromosome, thus contiguous window sampling could often miss the relevant signal. The training, validation, and test sets for the GCN had 224,555, 25,445, and 22,480 replicates respectively, with the latter being the same test set used to assess performance of the CNN. The training, validation and testing data were evenly balanced across the five categories.

Detecting introgressed loci

We used the *Drosophila* simulations generated for (Ray *et al.* 2023) as data for a 3-class classifier problem to benchmark each networks' ability to detect the presence and direction of introgression within a genomic region. In brief, we simulated a pulse migration event from *D.*

828 *simulans* to *D. sechellia* at a random time sampled from $U(0.3, 0.5) \times T$, where T is the
829 population split time inferred by bootstrapped `∂a∂i` (Gutenkunst *et al.* 2010) replicate inferences
830 on *D. simulans* inbred and *D. sechellia* wild-caught genomes. In contrast to the parameterization
831 simulated by Ray et al. (2023) which used more recent introgression events, we chose the
832 introgression time distribution to be $U(0.3, 0.5) \times T$ to make the problem a more difficult
833 benchmark (because older introgression events have introgressed haplotypes that are less
834 diverged between the donor and recipient population, and also that have had more time to be
835 broken up by recombination events after the introgression event).
836
837 For this task genotype matrices are formatted in the shape of (*samples*, *populations*, *individuals*,
838 *polymorphisms*). Data was padded to 665 polymorphisms or the maximum observed number of
839 polymorphisms in simulated replicates. The median number of trees returned by RELATE
840 (computed from training data) was 68 with a standard deviation of 11.56. For the GCN, the
841 training and validation sets had 232,050 and 25,950 replicates respectively. For the CNN, the
842 training and validation sets had 244,740 and 13,260 replicates respectively. The test set used to
843 compare the methods contained 12,900 replicates, and the training, validation, and test sets were
844 all class-balanced.
845
846 *Demographic inference*
847 As in Flagel *et al.* (2019) we tested the ability of our networks to perform demographic inference
848 by simulating a 3-epoch demographic model with 5 parameters: the present-day population size
849 (N_0 which was drawn from $U(100, 4\times10^4)$), the time of the most recent population size change
850 ($T_1 \sim U(100, 3500)$), the population size during the middle epoch ($N_1 \sim U(100, 5000)$), the time of

the ancient population size change (T_2 , which was set to T_1 plus a value drawn from $U(1, 3500)$), and the ancestral population size ($N_2 \sim U(100, 2\times10^4)$). These data were simulated using *ms* and the corresponding scripts can be found at https://github.com/flag0010/pop_gen_cnn/tree/master/demography. For the CNN, the training / validation had 94,036 and 5,164 replicates respectively.

The data was padded to 652 polymorphisms or the maximum observed number of polymorphisms in the simulated replicates. Target values were log-scaled and standardized prior to training and testing, with the validation and testing data being standardized according to the mean and standard deviation of the log-scaled training data. For the GCN, the training and validation sets had 82,820 and 9,168 replicates respectively. The median number of trees in sequence was 53 with a standard deviation of 37.0. The test set used to compare the two methods contained 3,995.

DATA AND CODE AVAILABILITY

All code associated with this work can be found at https://github.com/SchriderLab/TreeSeqPopGenInference. Model weights can be downloaded from https://drive.google.com/drive/folders/1WRf8pecfRavOQjmZF7Otr4lXbcouVunm?usp=sharing .

ACKNOWLEDGMENTS

The authors thank Will Booker for comments on the manuscript, and members of the Kern and Ralph Labs for feedback on preliminary data. LSW was supported by NIH award R01AI153523,

DDR was supported by NIH award R01HG010774, and DRS was supported by NIH award R35GM138286.

REFERENCES

Adrion J. R., J. G. Galloway, and A. D. Kern, 2020 Predicting the Landscape of Recombination Using Deep Learning. Mol. Biol. Evol. 37: 1790–1808. https://doi.org/10.1093/molbev/msaa038

Agarap A. F., 2018 Deep learning using rectified linear units (relu). ArXiv Prepr. ArXiv180308375.

Arnab S. P., M. R. Amin, and M. DeGiorgio, 2022 Uncovering footprints of natural selection through time-frequency analysis of genomic summary statistics. 2022.10.05.510997.

Auton A., and G. McVean, 2007 Recombination rate estimation in the presence of hotspots. Genome Res. 17: 1219–1227. https://doi.org/10.1101/gr.6386707

Auton A., G. R. Abecasis, D. M. Altshuler, R. M. Durbin, G. R. Abecasis, *et al.*, 2015 A global reference for human genetic variation. Nature 526: 68–74. https://doi.org/10.1038/nature15393

Ba J. L., J. R. Kiros, and G. E. Hinton, 2016 Layer Normalization

Battey C., P. L. Ralph, and A. D. Kern, 2020 Predicting geographic location from genetic variation with deep neural networks, (P. J. Wittkopp, M. Nordborg, C. Huber, A. Harpak, and L. Speidel, Eds.). eLife 9: e54507. https://doi.org/10.7554/eLife.54507

Battey C. J., G. C. Coffing, and A. D. Kern, 2021 Visualizing population structure with variational autoencoders. G3 GenesGenomesGenetics 11: jkaa036. https://doi.org/10.1093/g3journal/jkaa036

Beaumont M. A., W. Zhang, and D. J. Balding, 2002 Approximate Bayesian computation in population genetics. Genetics 162: 2025–2035. https://doi.org/10.1093/genetics/162.4.2025

Berg J. J., and G. Coop, 2015 A Coalescent Model for a Sweep of a Unique Standing Variant. Genetics 201: 707–725. https://doi.org/10.1534/genetics.115.178962

Booker W. W., D. D. Ray, and D. R. Schrider, 2023 This population does not exist: learning the distribution of evolutionary histories with generative adversarial networks. Genetics iyad063. https://doi.org/10.1093/genetics/iyad063

Brody S., U. Alon, and E. Yahav, 2022 How Attentive are Graph Attention Networks?

Browning S. R., and B. L. Browning, 2015 Accurate Non-parametric Estimation of Recent Effective Population Size from Segments of Identity by Descent. Am. J. Hum. Genet. 97: 404–418. https://doi.org/10.1016/j.ajhg.2015.07.012

Caldas I. V., A. G. Clark, and P. W. Messer, 2022 Inference of selective sweep parameters through supervised learning. 2022.07.19.500702.

Chan A. H., P. A. Jenkins, and Y. S. Song, 2012 Genome-Wide Fine-Scale Recombination Rate Variation in Drosophila melanogaster. PLOS Genet. 8: e1003090. https://doi.org/10.1371/journal.pgen.1003090

Chan J., V. Perrone, J. Spence, P. Jenkins, S. Mathieson, *et al.*, 2018 A Likelihood-Free Inference Framework for Population Genetic Data using Exchangeable Neural Networks, in *Advances in Neural Information Processing Systems*, Curran Associates, Inc.

Cho K., B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, *et al.*, 2014 Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation

DeGiorgio M., and Z. A. Szpiech, 2022 A spatially aware likelihood test to detect sweeps from

920 haplotype distributions. PLOS Genet. 18: e1010134.
921 https://doi.org/10.1371/journal.pgen.1010134
922 Erhan D., C. Szegedy, A. Toshev, and D. Anguelov, 2013 Scalable Object Detection using Deep
923 Neural Networks
924 Excoffier L., N. Marchi, D. A. Marques, R. Matthey-Doret, A. Gouy, *et al.*, 2021 fastsimcoal2:
925 demographic inference under complex evolutionary scenarios. Bioinformatics 37: 4882–
926 4885. https://doi.org/10.1093/bioinformatics/btab468
927 Fay J. C., and C. I. Wu, 2000 Hitchhiking under positive Darwinian selection. Genetics 155:
928 1405–1413.
929 Ferrer-Admetlla A., M. Liang, T. Korneliussen, and R. Nielsen, 2014 On detecting incomplete
930 soft or hard selective sweeps using haplotype structure. Mol. Biol. Evol. 31: 1275–1291.
931 https://doi.org/10.1093/molbev/msu077
932 Flagel L., Y. Brandvain, and D. R. Schrider, 2019 The Unreasonable Effectiveness of
933 Convolutional Neural Networks in Population Genetic Inference. Mol. Biol. Evol. 36:
934 220–238. https://doi.org/10.1093/molbev/msy224
935 Fu Y. X., and W. H. Li, 1993 Statistical Tests of Neutrality of Mutations. Genetics 133: 693–
936 709.
937 Ganin Y., and V. Lempitsky, 2015 Unsupervised Domain Adaptation by Backpropagation
938 Gao F., C. Ming, W. Hu, and H. Li, 2016 New Software for the Fast Estimation of Population
939 Recombination Rates (FastEPRR) in the Genomic Era. G3 GenesGenomesGenetics 6:
940 1563–1571. https://doi.org/10.1534/g3.116.028233
941 Garud N. R., P. W. Messer, E. O. Buzbas, and D. A. Petrov, 2015 Recent Selective Sweeps in
942 North American Drosophila melanogaster Show Signatures of Soft Sweeps. PLOS Genet.

943 11: e1005004. https://doi.org/10.1371/journal.pgen.1005004
944 Girshick R., 2015 Fast R-CNN
945 Gower G., P. I. Picazo, M. Fumagalli, and F. Racimo, 2021 Detecting adaptive introgression in
946 human evolution using convolutional neural networks. eLife 10: e64669.
947 https://doi.org/10.7554/eLife.64669
948 Gu J., Z. Wang, J. Kuen, L. Ma, A. Shahroudy, *et al.*, 2018 Recent advances in convolutional
949 neural networks. Pattern Recognit. 77: 354–377.
950 https://doi.org/10.1016/j.patcog.2017.10.013
951 Gutenkunst R. N., R. D. Hernandez, S. H. Williamson, and C. D. Bustamante, 2009 Inferring the
952 Joint Demographic History of Multiple Populations from Multidimensional SNP
953 Frequency Data. PLOS Genet. 5: e1000695.
954 https://doi.org/10.1371/journal.pgen.1000695
955 Gutenkunst R., R. Hernandez, S. Williamson, and C. Bustamante, 2010 Diffusion
956 Approximations for Demographic Inference: DaDi. Nat. Preced. 1–1.
957 https://doi.org/10.1038/npre.2010.4594.1
958 Haller B. C., J. Galloway, J. Kelleher, P. W. Messer, and P. L. Ralph, 2019 Tree-sequence
959 recording in SLiM opens new horizons for forward-time simulation of whole genomes.
960 Mol. Ecol. Resour. 19: 552–566. https://doi.org/10.1111/1755-0998.12968
961 Haller B. C., and P. W. Messer, 2022 SLiM 4: Multispecies Eco-Evolutionary Modeling. Am.
962 Nat. E000–E000. https://doi.org/10.1086/723601
963 He K., X. Zhang, S. Ren, and J. Sun, 2015 Deep Residual Learning for Image Recognition
964 Hejase H. A., Z. Mo, L. Campagna, and A. Siepel, 2022 A Deep-Learning Approach for
965 Inference of Selective Sweeps from the Ancestral Recombination Graph. Mol. Biol. Evol.

966 39: msab332. https://doi.org/10.1093/molbev/msab332
967 Hermisson J., and P. S. Pennings, 2005 Soft Sweeps. Genetics 169: 2335–2352.
968 https://doi.org/10.1534/genetics.104.036947
969 Hermisson J., and P. S. Pennings, 2017 Soft sweeps and beyond: understanding the patterns and
970 probabilities of selection footprints under rapid adaptation. Methods Ecol. Evol. 8: 700–
971 716. https://doi.org/10.1111/2041-210X.12808
972 Hill W. G., and A. Robertson, 1966 The effect of linkage on limits to artificial selection. Genet.
973 Res. 8: 269–294.
974 Hochreiter S., and J. Schmidhuber, 1997 Long Short-Term Memory. Neural Comput. 9: 1735–
975 1780. https://doi.org/10.1162/neco.1997.9.8.1735
976 Hudson R. R., M. Slatkin, and W. P. Maddison, 1992 Estimation of Levels of Gene Flow from
977 DNA Sequence Data. Genetics 132: 583–589.
978 Hudson R. R., K. Bailey, D. Skarecky, J. Kwiatowski, and F. J. Ayala, 1994 Evidence for
979 positive selection in the superoxide dismutase (Sod) region of Drosophila melanogaster.
980 Genetics 136: 1329–1340. https://doi.org/10.1093/genetics/136.4.1329
981 Hudson R. R., 2001 Two-locus sampling distributions and their application. Genetics 159: 1805–
982 1817. https://doi.org/10.1093/genetics/159.4.1805
983 Hudson R. R., 2002 Generating samples under a Wright–Fisher neutral model of genetic
984 variation. Bioinformatics 18: 337–338. https://doi.org/10.1093/bioinformatics/18.2.337
985 Huson D. H., T. Klöpper, P. J. Lockhart, and M. A. Steel, 2005 Reconstruction of Reticulate
986 Networks from Gene Trees, pp. 233–249 in *Research in Computational Molecular*
987 *Biology*, Lecture Notes in Computer Science. edited by Miyano S., Mesirov J., Kasif S.,
988 Istrail S., Pevzner P. A., *et al.* Springer, Berlin, Heidelberg.

Ioffe S., and C. Szegedy, 2015 Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Kamm J., J. Terhorst, R. Durbin, and Y. S. Song, 2020 Efficiently inferring the demographic history of many populations with allele count data. J. Am. Stat. Assoc. 115: 1472–1487. https://doi.org/10.1080/01621459.2019.1635482

Kaplan N. L., R. R. Hudson, and C. H. Langley, 1989 The "hitchhiking effect" revisited. Genetics 123: 887–899. https://doi.org/10.1093/genetics/123.4.887

Karasov T., P. W. Messer, and D. A. Petrov, 2010 Evidence that Adaptation in Drosophila Is Not Limited by Mutation at Single Sites. PLOS Genet. 6: e1000924. https://doi.org/10.1371/journal.pgen.1000924

Kelleher J., A. M. Etheridge, and G. McVean, 2016 Efficient Coalescent Simulation and Genealogical Analysis for Large Sample Sizes. PLOS Comput. Biol. 12: e1004842. https://doi.org/10.1371/journal.pcbi.1004842

Kelleher J., K. R. Thornton, J. Ashander, and P. L. Ralph, 2018 Efficient pedigree recording for fast population genetics simulation. PLOS Comput. Biol. 14: e1006581. https://doi.org/10.1371/journal.pcbi.1006581

Kelleher J., Y. Wong, A. W. Wohns, C. Fadil, P. K. Albers, *et al.*, 2019 Inferring whole-genome histories in large population datasets. Nat. Genet. 51: 1330–1338. https://doi.org/10.1038/s41588-019-0483-y

Kelly J. K., 1997 A test of neutrality based on interlocus associations. Genetics 146: 1197–1206. https://doi.org/10.1093/genetics/146.3.1197

Kern A. D., and D. R. Schrider, 2016 Discoal: flexible coalescent simulations with selection. Bioinformatics 32: 3839–3841. https://doi.org/10.1093/bioinformatics/btw556

Kern A. D., and D. R. Schrider, 2018 diploS/HIC: An Updated Approach to Classifying Selective Sweeps. G3 GenesGenomesGenetics 8: 1959–1970. https://doi.org/10.1534/g3.118.200262

Kim Y., and R. Nielsen, 2004 Linkage disequilibrium as a signature of selective sweeps. Genetics 167: 1513–1524. https://doi.org/10.1534/genetics.103.025387

Kingma D. P., and J. Ba, 2017 Adam: A Method for Stochastic Optimization

Kipf T. N., and M. Welling, 2017 Semi-Supervised Classification with Graph Convolutional Networks

Korfmann K., O. E. Gaggiotti, and M. Fumagalli, 2023a Deep Learning in Population Genetics. Genome Biol. Evol. 15: evad008. https://doi.org/10.1093/gbe/evad008

Korfmann K., T. Sellinger, F. Freund, M. Fumagalli, and A. Tellier, 2023b Simultaneous Inference of Past Demography and Selection from the Ancestral Recombination Graph under the Beta Coalescent. 2022.09.28.508873.

Lauterbur M. E., K. Munch, and D. Enard, 2022 Versatile detection of diverse selective sweeps with Flex-sweep. 2022.11.15.516494.

LeCun Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, *et al.*, 1989 Backpropagation Applied to Handwritten Zip Code Recognition. Neural Comput. 1: 541–551. https://doi.org/10.1162/neco.1989.1.4.541

LeCun Y., L. Bottou, Y. Bengio, and P. Haffner, 1998 Gradient-based learning applied to document recognition. Proc. IEEE 86: 2278–2324.

Li H., and R. Durbin, 2011 Inference of Human Population History From Whole Genome Sequence of A Single Individual. Nature 475: 493–496. https://doi.org/10.1038/nature10231

Li Z., F. Liu, W. Yang, S. Peng, and J. Zhou, 2022 A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. IEEE Trans. Neural Netw. Learn. Syst. 33: 6999–7019. https://doi.org/10.1109/TNNLS.2021.3084827

Lin K., H. Li, C. Schlötterer, and A. Futschik, 2011 Distinguishing Positive Selection From Neutral Evolution: Boosting the Performance of Summary Statistics. Genetics 187: 229–244. https://doi.org/10.1534/genetics.110.122614

Mahmoudi A., J. Koskela, J. Kelleher, Y. Chan, and D. Balding, 2022 Bayesian inference of ancestral recombination graphs. PLOS Comput. Biol. 18: e1009960. https://doi.org/10.1371/journal.pcbi.1009960

Mo Z., and A. Siepel, 2023 Domain-adaptive neural networks improve supervised machine learning based on simulated population genetic data. 2023.03.01.529396.

Mughal M. R., and M. DeGiorgio, 2019 Localizing and Classifying Adaptive Targets with Trend Filtered Regression. Mol. Biol. Evol. 36: 252–270. https://doi.org/10.1093/molbev/msy205

Mughal M. R., H. Koch, J. Huang, F. Chiaromonte, and M. DeGiorgio, 2020 Learning the properties of adaptive regions with functional data analysis. PLOS Genet. 16: e1008896. https://doi.org/10.1371/journal.pgen.1008896

Nei M., and W. H. Li, 1979 Mathematical model for studying genetic variation in terms of restriction endonucleases. Proc. Natl. Acad. Sci. 76: 5269–5273. https://doi.org/10.1073/pnas.76.10.5269

Nei M., and F. Tajima, 1981 DNA Polymorphism Detectable by Restriction Endonucleases. Genetics 97: 145–163.

Paszke A., S. Gross, F. Massa, A. Lerer, J. Bradbury, *et al.*, 2019 PyTorch: An Imperative Style,

High-Performance Deep Learning Library

Pavlidis P., J. D. Jensen, and W. Stephan, 2010 Searching for footprints of positive selection in whole-genome SNP data from nonequilibrium populations. Genetics 185: 907–922. https://doi.org/10.1534/genetics.110.116459

Pearson A., and R. Durbin, 2023 Local Ancestry Inference for Complex Population Histories. 2023.03.06.529121.

Perron L., and V. Furnon, 2019 OR-Tools

Pritchard J. K., M. T. Seielstad, A. Perez-Lezaun, and M. W. Feldman, 1999 Population growth of human Y chromosomes: a study of Y chromosome microsatellites. Mol. Biol. Evol. 16: 1791–1798. https://doi.org/10.1093/oxfordjournals.molbev.a026091

Przeworski M., G. Coop, and J. D. Wall, 2005 The signature of positive selection on standing genetic variation. Evol. Int. J. Org. Evol. 59: 2312–2323.

Pudlo P., J.-M. Marin, A. Estoup, J.-M. Cornuet, M. Gautier, *et al.*, 2016 Reliable ABC model choice via random forests. Bioinformatics 32: 859–866. https://doi.org/10.1093/bioinformatics/btv684

Pybus M., P. Luisi, G. M. Dall'Olio, M. Uzkudun, H. Laayouni, *et al.*, 2015 Hierarchical boosting: a machine-learning framework to detect and classify hard selective sweeps in human populations. Bioinformatics 31: 3946–3952. https://doi.org/10.1093/bioinformatics/btv493

Ralph P., K. Thornton, and J. Kelleher, 2020 Efficiently Summarizing Relationships in Large Samples: A General Duality Between Statistics of Genealogies and Genomes. Genetics 215: 779–797. https://doi.org/10.1534/genetics.120.303253

Rasmussen M. D., M. J. Hubisz, I. Gronau, and A. Siepel, 2014 Genome-Wide Inference of

Ancestral Recombination Graphs. PLOS Genet. 10: e1004342. https://doi.org/10.1371/journal.pgen.1004342

Ray D. D., L. Flagel, and D. R. Schrider, 2023 IntroUNET: identifying introgressed alleles via semantic segmentation. 2023.02.07.527435.

Raynal L., J.-M. Marin, P. Pudlo, M. Ribatet, C. P. Robert, *et al.*, 2019 ABC random forests for Bayesian parameter inference. Bioinforma. Oxf. Engl. 35: 1720–1728. https://doi.org/10.1093/bioinformatics/bty867

Reddi V. J., C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, *et al.*, 2020 MLPerf Inference Benchmark

Ronen R., N. Udpa, E. Halperin, and V. Bafna, 2013 Learning Natural Selection from the Site Frequency Spectrum. Genetics 195: 181–193. https://doi.org/10.1534/genetics.113.152587

Rosenzweig B. K., J. B. Pease, N. J. Besansky, and M. W. Hahn, 2016 Powerful methods for detecting introgressed regions from population genomic data. Mol. Ecol. 25: 2387–2397. https://doi.org/10.1111/mec.13610

Sabeti P. C., D. E. Reich, J. M. Higgins, H. Z. P. Levine, D. J. Richter, *et al.*, 2002 Detecting recent positive selection in the human genome from haplotype structure. Nature 419: 832–837. https://doi.org/10.1038/nature01140

Sanchez T., J. Cury, G. Charpiat, and F. Jay, 2021 Deep learning for population size history inference: Design, comparison and combination with approximate Bayesian computation. Mol. Ecol. Resour. 21: 2645–2660. https://doi.org/10.1111/1755-0998.13224

Santiago E., I. Novo, A. F. Pardiñas, M. Saura, J. Wang, *et al.*, 2020 Recent Demographic History Inferred by High-Resolution Analysis of Linkage Disequilibrium. Mol. Biol.

Evol. 37: 3642–3653. https://doi.org/10.1093/molbev/msaa169

Schrider D. R., and A. D. Kern, 2016 S/HIC: Robust Identification of Soft and Hard Sweeps Using Machine Learning. PLOS Genet. 12: e1005928. https://doi.org/10.1371/journal.pgen.1005928

Schrider D. R., and A. D. Kern, 2017 Soft Sweeps Are the Dominant Mode of Adaptation in the Human Genome. Mol. Biol. Evol. 34: 1863–1877. https://doi.org/10.1093/molbev/msx154

Schrider D. R., and A. D. Kern, 2018 Supervised Machine Learning for Population Genetics: A New Paradigm. Trends Genet. TIG 34: 301–312. https://doi.org/10.1016/j.tig.2017.12.005

Schrider D. R., J. Ayroles, D. R. Matute, and A. D. Kern, 2018 Supervised machine learning reveals introgressed loci in the genomes of Drosophila simulans and D. sechellia. PLOS Genet. 14: e1007341. https://doi.org/10.1371/journal.pgen.1007341

Sheehan S., and Y. S. Song, 2016 Deep Learning for Population Genetic Inference. PLoS Comput. Biol. 12: e1004845. https://doi.org/10.1371/journal.pcbi.1004845

Shriner D., D. C. Nickle, M. A. Jensen, and J. I. Mullins, 2003 Potential impact of recombination on sitewise approaches for detecting positive natural selection. Genet. Res. 81: 115–121. https://doi.org/10.1017/S0016672303006128

Smith J. M., and J. Haigh, 1974 The hitch-hiking effect of a favourable gene. Genet. Res. 23: 23–35. https://doi.org/10.1017/S0016672300014634

Smith C. C. R., S. Tittes, P. L. Ralph, and A. D. Kern, 2023 Dispersal inference from population genetic variation using a convolutional neural network. Genetics iyad068. https://doi.org/10.1093/genetics/iyad068

Speidel L., M. Forest, S. Shi, and S. R. Myers, 2019 A method for genome-wide genealogy estimation for thousands of samples. Nat. Genet. 51: 1321–1329. https://doi.org/10.1038/s41588-019-0484-x

Stephan W., 2019 Selective Sweeps. Genetics 211: 5–13. https://doi.org/10.1534/genetics.118.301319

Tajima F., 1989 Statistical Method for Testing the Neutral Mutation Hypothesis by DNA Polymorphism. Genetics 123: 585–595.

Tavare S., D. J. Balding, R. C. Griffiths, and P. Donnelly, 1997 Inferring Coalescence Times from DNA Sequence Data. Genetics 145: 505. https://doi.org/10.1093/genetics/145.2.505

Terhorst J., J. A. Kamm, and Y. S. Song, 2017 Robust and scalable inference of population history from hundreds of unphased whole-genomes. Nat. Genet. 49: 303–309. https://doi.org/10.1038/ng.3748

Vapnik V., 2006 *Estimation of Dependences Based on Empirical Data*. Springer New York, New York, NY.

Veličković P., G. Cucurull, A. Casanova, A. Romero, P. Liò, *et al.*, 2018 Graph Attention Networks

Voight B. F., S. Kudaravalli, X. Wen, and J. K. Pritchard, 2006 A Map of Recent Positive Selection in the Human Genome. PLoS Biol. 4: e72. https://doi.org/10.1371/journal.pbio.0040072

Wang Z., J. Wang, M. Kourakos, N. Hoang, H. H. Lee, *et al.*, 2021 Automatic inference of demographic parameters using generative adversarial networks. Mol. Ecol. Resour. 21: 2689–2705. https://doi.org/10.1111/1755-0998.13386

Watterson G. A., 1975 On the number of segregating sites in genetical models without

recombination. Theor. Popul. Biol. 7: 256–276. https://doi.org/10.1016/0040-5809(75)90020-9

Whitehouse L. S., and D. R. Schrider, 2022 Timesweeper: Accurately Identifying Selective Sweeps Using Population Genomic Time Series. 2022.07.06.499052.

Whitehouse L. S., and D. R. Schrider, 2023 Timesweeper: accurately identifying selective sweeps using population genomic time series. Genetics 224: iyad084. https://doi.org/10.1093/genetics/iyad084

Wiuf C., and J. Hein, 1999 Recombination as a Point Process along Sequences. Theor. Popul. Biol. 55: 248–259. https://doi.org/10.1006/tpbi.1998.1403

Yelmen B., A. Decelle, L. Ongaro, D. Marnetto, C. Tallec, *et al.*, 2021 Creating artificial human genomes using generative neural networks. PLOS Genet. 17: e1009303. https://doi.org/10.1371/journal.pgen.1009303

Zhang B. C., A. Biddanda, Á. F. Gunnarsson, F. Cooper, and P. F. Palamara, 2023 Biobank-scale inference of ancestral recombination graphs enables genealogical analysis of complex traits. Nat. Genet. 55: 768–776. https://doi.org/10.1038/s41588-023-01379-x

Zhou J., G. Cui, S. Hu, Z. Zhang, C. Yang, *et al.*, 2020 Graph neural networks: A review of methods and applications. AI Open 1: 57–81.