dnadna: A DEEP LEARNING FRAMEWORK FOR POPULATION GENETIC INFERENCE

Théophile Sanchez, Erik Madison Bray, Pierre Jobic, Jérémy Guez,
Anne-Catherine Letournel, Guillaume Charpiat, Jean Cury, Flora Jay

DNADNA: A DEEP LEARNING FRAMEWORK FOR POPULATION GENETIC INFERENCE

A PREPRINT

Théophile Sanchez [1*], **Erik Madison Bray** [1*],
Pierre Jobic [1,2], **Jérémy Guez** [1,3], **Anne-Catherine Letournel** [1], **Guillaume Charpiat** [1],
Jean Cury [1+], **Flora Jay** [1+]

[1] Université Paris-Saclay, CNRS UMR 9015, INRIA,
Laboratoire Interdisciplinaire des Sciences du Numérique, 91400, Orsay, France.
[2] ENS Paris-Saclay France.
[3] UMR7206 Eco-Anthropologie, Muséum National d'Histoire Naturelle, CNRS, Université de Paris, Paris, France.
[4] SEED, U1284, INSERM, Université de Paris, Paris, France

[*] These authors contributed equally
[+] Corresponding authors. These authors contributed equally.
Corresponding authors:
flora.jay@lri.fr, jean.cury@normalesup.org

ABSTRACT

We present `dnadna`, a flexible python-based software for deep learning inference in population genetics. It is task-agnostic and aims at facilitating the development, reproducibility, dissemination, and reusability of neural networks designed for population genetic data.

`dnadna` defines multiple user-friendly workflows. First, users can implement new architectures and tasks, while benefiting from `dnadna` utility functions, training procedure and test environment, which saves time and decreases the likelihood of bugs. Second, the implemented networks can be re-optimized based on user-specified training sets and/or tasks. Newly implemented architectures and pretrained networks are easily shareable with the community for further benchmarking or other applications. Finally, users can apply pretrained networks in order to predict evolutionary history from alternative real or simulated genetic datasets, without requiring extensive knowledge in deep learning or coding in general.

`dnadna` comes with a peer-reviewed, exchangeable neural network, allowing demographic inference from SNP data, that can be used directly or retrained to solve other tasks. Toy networks are also available to ease the exploration of the software, and we expect that the range of available architectures will keep expanding thanks to community contributions.

Availability: `dnadna` repository is available at `gitlab.com/mlgenetics/dnadna` and its associated documentation at `mlgenetics.gitlab.io/dnadna/`.

Keywords population genetics · genomic data · deep neural networks · supervised learning · software

1 Introduction

In the past decade, many deep learning methods have been implemented with the hope of facilitating complex data analyses and information discovery in various applied fields. In the context of biological research, they have been increasingly used to derive novel insights from high-dimensional biological data (Angermueller *et al.*, 2016). Population genetics did not escape this trend and deep learning based approaches are now flourishing. As reviewed by Sanchez *et al.* (2020), we distinguish two families of deep learning approaches for population genetic inference: those processing many summary statistics, with either fully connected or convolutional deep neural networks (e.g. Sheehan and Song,

2016; Mondal *et al.*, 2019; Xue *et al.*, 2021; Villanea and Schraiber, 2019; Montinaro *et al.*, 2021) and those based on 'raw' genetic data.

The latter, which are the main targets of `dnadna`, leverage deep neural networks to automatically construct informative features and bypass handcrafted summary statistics. Since 2008, a wide range of networks (fully-connected, convolutional, exchangeable convolutional, recurrent, adversarial, restricted Boltzmann machines, variational autoencoder) have been applied to multiple population genetic tasks, namely the inference of recombination or mutation rates (Chan *et al.*, 2018; Adrion *et al.*, 2020b; Burger *et al.*, 2022), various types of selection (Flagel *et al.*, 2019; Deelder *et al.*, 2021; Torada *et al.*, 2019; Gower *et al.*, 2021; Isildak *et al.*, 2021; Qin *et al.*, 2022), species delimitation (Derkarabetian *et al.*, 2019; Perez *et al.*, 2022), local or global introgression and ancestry (Flagel *et al.*, 2019; Montserrat *et al.*, 2020; Karim *et al.*, 2020; Wang *et al.*, 2021; Fonseca *et al.*, 2021; Meisner and Albrechtsen, 2022), past effective population sizes (Sanchez *et al.*, 2020; Wang *et al.*, 2021), geographic location (Battey *et al.*, 2020) and the visualization and/or generation of individual genomes (Yelmen *et al.*, 2021; Battey *et al.*, 2021; Montserrat *et al.*, 2019; Chen *et al.*, 2020), to name a few.

Previous studies have made their implementations available at least for reproducibility and sometimes with a specific effort for re-usability. Even so, each of them focuses on a specific network for a specific task. These implementations can usually be adapted to newly simulated datasets, and some hyperparameters can be modified. However, this requires careful reading and understanding of the code, and its direct modification since many options are hard-coded. This is not only error-prone, but also leads to a very rapid divergence of code between parallel projects, accompanied by complex maintenance. We know the importance of having flexible and rigorous tools for the community, and this was recently demonstrated by the emergence of `stdpopsim`, a standardized library for population genetic simulations that provides thoroughly checked demographic scenarios, corresponding to previously inferred species histories, and which allows contributions from many researchers (Adrion *et al.*, 2020a). In genomics a suite of tools, including `Janggu` (Kopp *et al.*, 2020), `AMBER` (Zhang *et al.*, 2021), `pysster` (Budach and Marsico, 2018), `Selene` (Chen *et al.*, 2019), `keras_dna` (Routhier *et al.*, 2020), `Kipoi` (Avsec *et al.*, 2019)), has been developed to facilitate one or several classical steps of deep learning pipelines (data formatting, network optimization, etc). However, none of them is able to handle population genetic datasets and tasks, as they focus instead on genomic tasks applied to single sequences, such as the prediction of transcription factor binding sites, histone modifications, or others (Routhier and Mozziconacci, 2022).

For these reasons we developed `dnadna`, Deep Neural Architectures for DNA, a comprehensive tool for population genetic inference. It is a task-agnostic software (in the context of population genetics) that aims at facilitating applications, development, distribution and exchanges around neural networks in the field. `dnadna` notably enables researchers to (1) develop new networks or re-use existing architectures, (2) train them for a given task and (3) share them in such a way that users can easily apply these trained networks to their own dataset.

2 Software

`dnadna` is a python-based software currently based on PyTorch. It implements the two main supervised machine learning tasks used for population genetic inference: regression and classification, and any tasks that are a mix of these. In particular, it already implements several neural networks that have been tested for inferring demographic and adaptive histories from genetic data. Pre-trained networks can be used directly on real and simulated genetic data for the prediction step. Implemented networks can also be retrained on new simulations (e.g. corresponding to another species or another evolutionary models) and/or to solve other tasks (e.g. classifying introgressed vs non-introgressed segments or inferring recombination). Finally, any user can implement new architectures and tasks, while benefiting from training procedure, test environment, routines that may be otherwise overlooked (such as proper preprocessing or efficient data loading), and the possibility to easily share a network to facilitate re-use and benchmarking.

Figure 1 provides an overview of `dnadna` steps, all of which are independent: (1) simulating, (2) preprocessing, (3) training, (4) predicting on test or real datasets. Note that the simulation step can be skipped if the user already possess a labelled dataset. Similarly, training can be skipped if the user reuses a pretrained network.

`dnadna` is accessible without coding knowledge thanks to its YAML configuration files which provide the user with a variety of options at each step of the process. We highlight here a few of the options for: (1a) generating simulations: name of predefined scenario to be simulated and its related parameters, such as number of individuals, number of replicates, mutation rate, demographic parameters; (1b) handling simulations: location on disk and filesystem layout; (2) preprocessing: initial data transformations, filtering values such as minimal number of sampled individuals or SNPs; (3a) architecture design: network name and related arguments (number of filters, layers, ...); (3b) training: loss functions, optimization and training hyperparameters (number of epochs, learning rate, batch size, optimizer name, ...); (3c) on-the-fly data transformations (subsampling, cropping, ...).

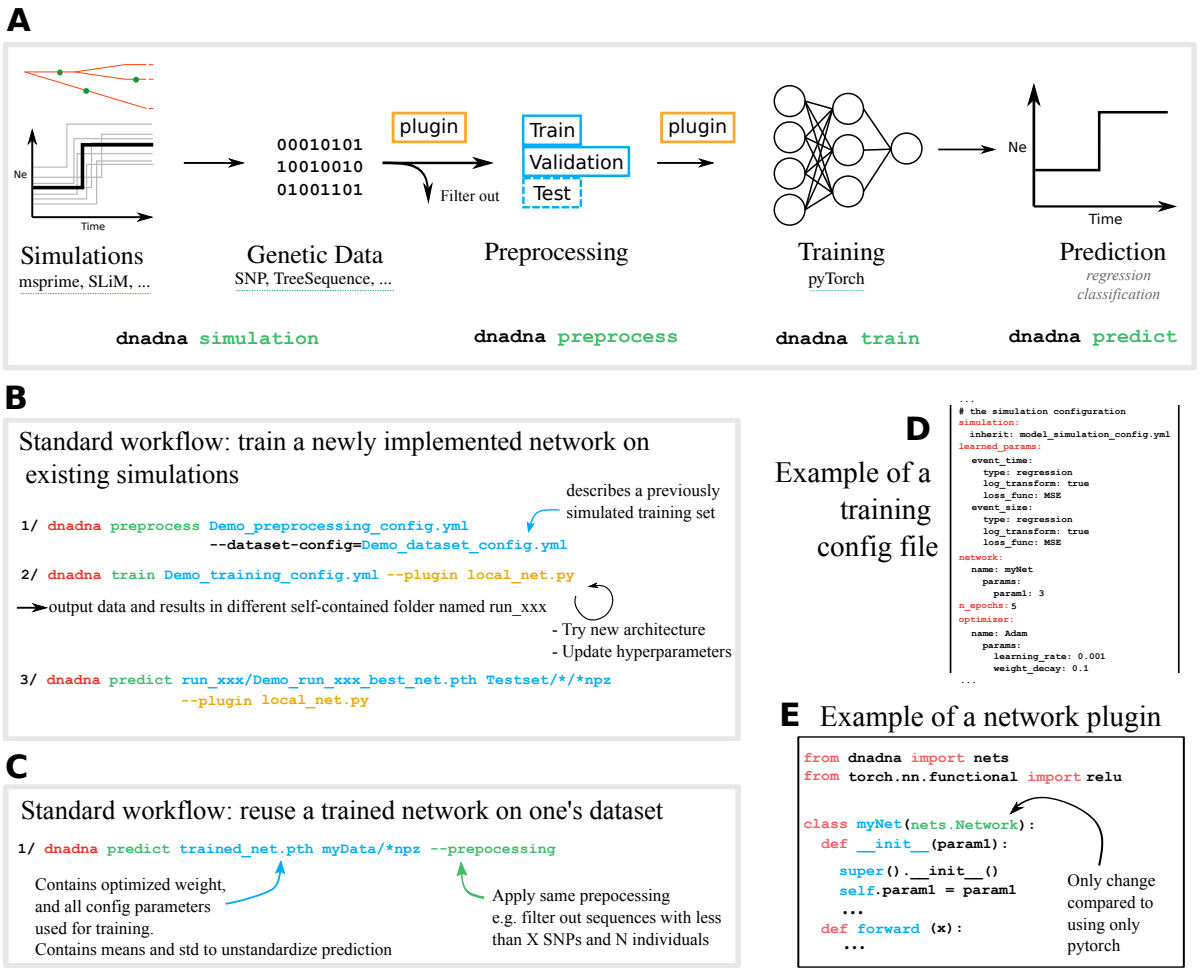

Figure 1: A: `dnadna` workflow and its corresponding commands. Each step could be done as a standalone. They are the classical steps required for any simulation-based supervised machine learning algorithm: (1) simulation of a large genetic dataset according to evolutionary scenarios and priors; genetic data type is not enforced and can be boolean (classical SNP data), integer (e.g. genotype data 0/1/2), or float (e.g. local density of SNPs as in Gower *et al.* (2021) or summary statistics along the genome as in Xue *et al.* (2021)) ; (2) preprocessing, mainly to filter out examples that do not fit minimal requirements and split the rest into train/validation/test sets; (3) training neural networks; (4) predicting on test or real datasets using optimized neural networks. B and C: illustration of two standard use cases of `dnadna`. D: extract of a training configuration file in YAML format. E: view of a plugin python file, that will be passed to `dnadna train`, where users can implement novel networks based on PyTorch.

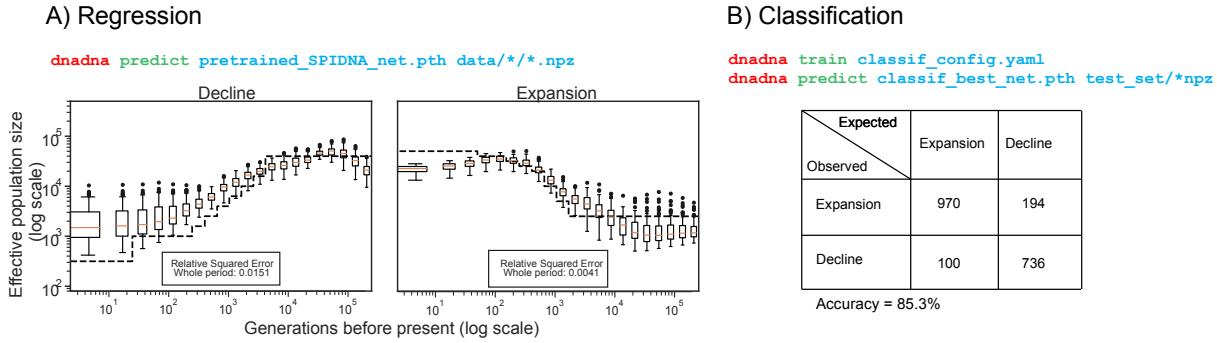

Figure 2: Illustration of two of `dnadna`'s use cases. A) Regression task with a pretrained network. Here we only need to use 'dnadna predict' since the network is already trained. Dotted lines denote true known histories, while boxplots indicate the population sizes predicted for 100 independent genomic regions at each time step. The estimates for a "complete" genome are given by the averaged predicted values. The error corresponds to the relative squared errors averaged over the whole time period. B) Classification task, following the quickstart tutorial 2. The network was trained on a toy dataset (2000 independent population samples split into training and validation) and tested on 2000 additional samples to discriminate whether the population underwent a decline or an expansion of its size. The contingency table shows that for this classification task, the accuracy is 85.3% on a test set. See `https://mlgenetics.gitlab.io/dnadna/tutorials.html` for details on both experiments.

Because each use-case has its own specificity, we developed a plugin system which allows users to adapt `dnadna`. For instance, users can implement new simulators, new data transformations or new networks, on top of what is already shipped with `dnadna`, without modifying the core of the code (see plugin documentation). By default, plugins need to be shared along with the optimized network, but they could also be merged into the code to benefit from maintenance, unit tests, and continuous integration to facilitate their adoption.

For the exhaustive and up-to-date list of parameters, options and detailed instruction for each step of the process, the reader should refer to the online documentation `https://mlgenetics.gitlab.io/dnadna/`.

3 Tutorial examples

We provide multiple tutorials that will continue to expand. These tutorials aim at showcasing the various use-cases of `dnadna`. It is not required to have coding knowledge for the user to perform similar tasks.

A first Quickstart Tutorial walks the user through the complete `dnadna` process from configuring and generating simulated genetic data, to running data pre-processing on the simulated dataset, and training a convolutional network to solve a regression task (here predicting the parameters of a two-step population size history) based on that dataset. A `msprime` Simulator Tutorial illustrates how to implement a simulation plugin, here based on `msprime` (Kelleher *et al.*, 2016), that can be integrated into `dnadna` (i.e. callable with `dnadna simulation` and outputting files in `dnadna` format). A second Quickstart tutorial proposes an alternative, where the user solves a classification task instead (discriminate between population declines and expansions). The accuracy obtained from training a classifier on a small dataset is approximately 85% and is presented in Figure 2B. A third Quickstart tutorial helps users who already have simulated data, to familiarize themselves with `dnadna`. It shows how to train a SPIDNA network on a provided dataset. Note that users could input non-binary data encoding any type of genetic markers or summary statistics. We also provide tutorials in the form of jupyter notebooks. A first one presents an alternative to the simulator tutorial: how few lines can be added into an existing `msprime` script to save the genetic data directly in the `dnadna` format. A second notebook guides the user to reconstruct a piecewise-constant population size history with 21-steps thanks to a pre-trained network, such as the SPIDNA network optimized by Sanchez *et al.* (2020). A sample of the predictions are shown and evaluated in Figure 2.

4 Implementation and distribution

`dnadna` is a Python (>=3.7) package with multiple dependencies, the main one being the open source machine learning library PyTorch. It has a command line interface and an API (`https://mlgenetics.gitlab.io/dnadna/api.html`). It is highly flexible thanks to a structured configuration file system based on YAML and JSON Schema.

`dnadna` is dual-licensed under the GNU Lesser General Public License (LGPLv3+) and the compatible CeCILL-C Free Software License Agreement (CECILL-C). Release 1.0 is available from PyPI at `https://pypi.org/project/dnadna/` and from Anaconda at `https://anaconda.org/mlgenetics/dnadna`. Docker images are available at `https://hub.docker.com/u/mlgenetics`.

For developers, we recommend cloning the GitLab repository (follow installation instructions). This also enables running the test suite via `pytest -v`. These tests are run automatically when contributing to the GitLab project thanks to the continuous integration pipeline (tests are run on both CPU and GPU).

5 Conclusion

`dnadna`'s aim is to help researchers to focus on their research project, be it the analysis of population genetic data or building new population genetic methods, while benefiting from good development practice (testing, continuous integration, documentation, etc.). Having a common interface, instead of having many parallel projects, will limit the presence of bugs in the code. We emphasize that `dnadna` can be used both by users and developers of population genetic inference methods, and we encourage a culture of user-developers.

With this paper, we release `dnadna` version 1.0, which is a stable version with all features required to develop, share and reuse a network, as described above. However, due to the complexity of developing deep learning methods for population genomics, and to the huge number of (hyper)parameters involved, some tasks or formats may not be implemented yet. We cannot anticipate all configurations, yet by continuing the development of the plugin system, we hope to cover as many cases as possible. In the future, we plan to make `dnadna` framework-agnostic (permitting networks built with TensorFlow or Keras in addition to PyTorch) thanks to the MMdnn library (`https://github.com/Microsoft/MMdnn`). Overall `dnadna` is a substantial software with lots of ambition. We are eager for the deep learning and population genetics communities to embrace it, provide feedback and contribute to it.

Acknowledgements

TAU GPU platform (Titanic) and Kepler for computing resources. ASARD team (LISN), SADL LRI. DIM One Health 2017 (number RPH17094JJP), Human Frontier Science Project (number RGY0075/2019), Paris-Saclay Center for Data Science 2.0 (IRS), CNRS-80Prime TransIA, ANR-20-CE45-0010-01 RoDAPoG for funding. Matteo Fumagalli, as well as all the participants of the Virtual School "Inference of demography and selection using full genome data" organized by the DFG SPP1819, for beta-testing. Susana Ribeiro for fruitful discussions regarding the manuscript.

References

Adrion, J. R. *et al.* (2020a). A community-maintained standard library of population genetic models. *eLife*, **9**, e54967.

Adrion, J. R. *et al.* (2020b). Predicting the Landscape of Recombination Using Deep Learning. *Molecular Biology and Evolution*, **37**(6), 1790–1808.

Angermueller, C. *et al.* (2016). Deep learning for computational biology. *Molecular Systems Biology*, **12**(7), 878.

Avsec, *et al.* (2019). The Kipoi repository accelerates community exchange and reuse of predictive models for genomics. *Nature Biotechnology*, **37**(6), 592–600.

Battey, C. *et al.* (2020). Predicting geographic location from genetic variation with deep neural networks. *eLife*, **9**, e54507.

Battey, C. J. *et al.* (2021). Visualizing population structure with variational autoencoders. *G3*, **11**(1), 1–11.

Budach, S. and Marsico, A. (2018). pysster: classification of biological sequences by learning sequence and structure motifs with convolutional neural networks. *Bioinformatics*, **34**(17), 3035–3037.

Burger, K. *et al.* (2022). Neural networks for self-adjusting mutation rate estimation when the recombination rate is unknown. *PLOS Computational Biology*, **18**, e1010407.

Chan, J. *et al.* (2018). A Likelihood-Free Inference Framework for Population Genetic Data using Exchangeable Neural Networks. In *Advances in Neural Information Processing Systems*, volume 31.

Chen, J. *et al.* (2020). Population-scale Genomic Data Augmentation Based on Conditional Generative Adversarial Networks. In *Proceedings of the 11th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, pages 1–6, Virtual Event USA. ACM.

Chen, K. M. *et al.* (2019). Selene: a PyTorch-based deep learning library for sequence data. *Nature methods*, **16**(4), 315–318.

Deelder, W. *et al.* (2021). Using deep learning to identify recent positive selection in malaria parasite sequence data. *Malaria Journal*, **20**(1), 270.

Derkarabetian, S. *et al.* (2019). A demonstration of unsupervised machine learning in species delimitation. *Molecular Phylogenetics and Evolution*, **139**, 106562.

Flagel, L. *et al.* (2019). The Unreasonable Effectiveness of Convolutional Neural Networks in Population Genetic Inference. *Molecular Biology and Evolution*, **36**(2), 220–238.

Fonseca, E. M. *et al.* (2021). Phylogeographic model selection using convolutional neural networks. *Molecular Ecology Resources*, **21**(8), 2661–2675.

Gower, G. *et al.* (2021). Detecting adaptive introgression in human evolution using convolutional neural networks. *eLife*, **10**, e64669.

Isildak, U. *et al.* (2021). Distinguishing between recent balancing selection and incomplete sweep using deep neural networks. *Molecular Ecology Resources*, **21**(8), 2706–2718.

Karim, M. R. *et al.* (2020). Convolutional Embedded Networks for Population Scale Clustering and Bio-ancestry Inferencing. *arXiv:1805.12218 [cs, q-bio, stat]*. arXiv: 1805.12218.

Kelleher, J. *et al.* (2016). Efficient Coalescent Simulation and Genealogical Analysis for Large Sample Sizes. *PLOS Computational Biology*, **12**(5), e1004842.

Kopp, W. *et al.* (2020). Deep learning for genomics using Janggu. *Nature Communications*, **11**(1), 3488.

Meisner, J. and Albrechtsen, A. (2022). Haplotype and population structure inference using neural networks in whole-genome sequencing data. *Genome Research*, **32**(8), 1542–1552.

Mondal, M. *et al.* (2019). Approximate Bayesian computation with deep learning supports a third archaic introgression in Asia and Oceania. *Nature Communications*, **10**(1).

Montinaro, F. *et al.* (2021). Revisiting the out of Africa event with a deep-learning approach. *The American Journal of Human Genetics*, **108**(11), 2037–2051.

Montserrat, D. M. *et al.* (2019). Class-Conditional VAE-GAN for Local-Ancestry Simulation. *arXiv:1911.13220*.

Montserrat, D. M. *et al.* (2020). LAI-Net: Local-Ancestry Inference with Neural Networks. *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1314–1318. arXiv: 2004.10377.

Perez, M. F. *et al.* (2022). Coalescent-based species delimitation meets deep learning: Insights from a highly fragmented cactus system. *Molecular Ecology Resources*, **22**(3), 1016–1028.

Qin, X. *et al.* (2022). Deciphering signatures of natural selection via deep learning. *Briefings in Bioinformatics*, **23**(5), bbac354.

Routhier, E. and Mozziconacci, J. (2022). Genomics enters the deep learning era. *PeerJ*, **10**, e13613. Publisher: PeerJ Inc.

Routhier, E. *et al.* (2020). Keras_dna: A Wrapper For Fast Implementation Of Deep Learning Models In Genomics. *Bioinformatics*, **37**(11), 1593–1594.

Sanchez, T. *et al.* (2020). Deep learning for population size history inference: Design, comparison and combination with approximate Bayesian computation. *Molecular Ecology Resources*, **21**(8), 2645–2660.

Sheehan, S. and Song, Y. S. (2016). Deep Learning for Population Genetic Inference. *PLOS Computational Biology*, **12**(3), e1004845.

Torada, L. *et al.* (2019). ImaGene: a convolutional neural network to quantify natural selection from genomic data. *BMC Bioinformatics*, **20**(S9), 337.

Villanea, F. A. and Schraiber, J. G. (2019). Multiple episodes of interbreeding between Neanderthal and modern humans. *Nature Ecology & Evolution*, **3**(1), 39–44.

Wang, Z. *et al.* (2021). Automatic inference of demographic parameters using generative adversarial networks. *Molecular Ecology Resources*, **21**(8).

Xue, A. T. *et al.* (2021). Discovery of Ongoing Selective Sweeps within Anopheles Mosquito Populations Using Deep Learning. *Molecular Biology and Evolution*, **38**(3), 1168–1183.

Yelmen, B. *et al.* (2021). Creating artificial human genomes using generative neural networks. *PLOS Genetics*, **17**(2), e1009303.

Zhang, Z. *et al.* (2021). An automated framework for efficiently designing deep convolutional neural networks in genomics. *Nature Machine Intelligence*, **3**(5), 392–400.