GENLINK: ANCESTRY INFERENCE WITH GNNS ON IBD GRAPHS FOR GENETICALLY SIMILAR POPULATIONS

Aleksei Shmelev[1,2], **Nikita Chervov**[1], **Dmitry Glyzin**[1],
Valery Ilinsky[3], **Alexander Rakitko**[1,4,5], **Vladimir Shchur**[1]

[1]HSE University, Moscow, Russia
[2]AXXX, Moscow, Russia
[3]Eligens SIA, Riga, Latvia
[4]Genotek Ltd, Moscow, Russia
[5]Genotek Center: AI in Personalized Medicine, ITMO University, Saint-Petersburg, Russia
vshchur@hse.ru

ABSTRACT

Graph Neural Networks (GNNs) have recently shown significant effectiveness in analyzing structured graph data across diverse domains. At the same time, accurate inference of ancestry from genetic data, especially among genetically similar populations, remains challenging due to internal complexity of the genetic relationships and high dimensionality of SNP data. To address these challenges, we propose a novel GNN-based method for inferring individual's ancestry from a graph that represents the genetic relatedness between individuals. Genetic relatedness between two individuals is measured according to shared identity-by-descent (IBD) segments, which are the segments of a genome inherited from a close common ancestor. In this context, the ancestry inference task is formalized as node classification on graphs. We present three key contributions. First, we advance the population genetics methodology with a unique GNN-based framework for ancestry inference for closely related populations. Second, we present a novel GNN architecture which improves training stability and predictive performance for ancestry inference on IBD graphs. Third, we demonstrate that augmenting the dataset with unlabeled vertices (individuals with unknown ancestry) significantly improves prediction scores, because message-passing in GNNs effectively propagates ancestry-related information throughout the network.

1 INTRODUCTION

Human populations have been shaped by migration, isolation, and admixture, producing complex patterns of genetic similarity across individuals. Inferring ancestry from genetic data is a central task in population genetics. Traditional ancestry inference methods estimate a person's origin by comparing their DNA to a reference panel of previously studied individuals used as references (Alexander et al., 2009). Such methods are effective when reference groups are well represented, but often struggle in highly admixed populations.

Advances in large-scale genotyping enabled accurate detection of identity-by-descent (IBD) segments — contiguous genomic regions inherited from a recent common ancestor (Browning & Browning, 2013). These segments arise because relatives share pieces of DNA that haven't been broken yet by recombination, the process that reshuffles genetic material. Such recombination does not occur uniformly. Certain regions, known as recombination hotspots (Li et al., 2022), are more likely to be broken. This leads shared segments to become shorter over time. As a result, shared IBD segments provide a signal of recent shared ancestry. Algorithms such as the templated positional Burrows–Wheeler transform (TPBWT), have made IBD detection both scalable and robust to phasing and genotyping errors in biobank-scale datasets (Freyman et al., 2021). These advances make it possible to represent large individual groups as graphs (IBD graphs), where individuals are nodes and edges encode genetic relatedness based on length of shared IBD segments.

Depending on the cohort and the quality of available metadata, an IBD graph may include individuals whose population labels are missing or unreliable, which naturally leads to settings where only a subset of nodes has known ancestry information. In this work, we formulate ancestry inference as a node classification problem on an IBD graph and apply graph neural networks (GNNs) to exploit the structure of genetic relatedness together with simple and interpretable node features. We focus on realistic scenarios in which population labels are only partially available and where individuals must be evaluated independently against an existing reference cohort, which requires a carefully designed and explainable validation protocol. To the best of our knowledge, closely related population methods for ancestry inference on IBD graphs have not been previously developed, which motivates our extensive empirical study with a broad range of strong baselines and ablation experiments. We show that retaining individuals without known population labels can be beneficial, since these nodes enrich the graph topology and allow ancestry related information to propagate more effectively through the network. In addition, we introduce a new GNN architecture, `GNNM`, which further improves performance by addressing training instability and pushing the accuracy of ancestry inference beyond standard GNN baselines in scenarios with incomplete labeling.

2 DATA PREPARATION

All preprocessing steps applied to the raw genotyping data, together with a detailed description of the datasets used in this study, are provided in Appendix A.

IBD-graph construction Given a set of n individuals, we quantify pairwise genetic relatedness by summing the total length of shared IBD segments. Formally, for individuals i and j, the genetic similarity is

$$\mathrm{W}(i,j) = \sum_{k=1}^{K_{ij}} L_k^{ij} \tag{1}$$

where K_{ij} is the number of IBD segments shared by i and j and L_k^{ij} is the length in centimorgans (cM) of the kth shared segment. We assume that only segments of length at least 6cM are meaningful and discard all shorter segments from the analysis, while additionally requiring that at least one shared segment has length of at least 8cM.

We construct a weighted undirected graph $G = (V, E, W)$. The vertex set $V = \{v_1, v_2, \ldots, v_n\}$ represents individuals. There is an edge with weight $W(i,j)$ between two individuals i and j if their cumulative shared IBD length exceeds 8 centimorgans. The graph is not necessarily complete or connected, moreover, in practice it would be sparse, because most individuals would be unrelated to each other. The maximal possible value of genetic relatedness is approximately 6600 cM, which corresponds to twins or to the same person and matches the total length of the human genome. We also removed siblings (edge weight greater than 300 cM) from the analysis before passing graph to any model, since classifying siblings is a trivial task.

Each individual carries an initial population annotation that may contain errors. These annotations arise from self reporting in which individuals state their nationality or ethnic group during genetic testing. Because such sources can contain inconsistencies there is a need for robust methods that tolerate noisy labels.

Feature construction The initial IBD graph does not contain any node features since each vertex represents an individual with an assigned population label and edges carry weights that reflect genetic similarity. Because running a graph neural network requires feature representations for all nodes, it becomes necessary to construct such features explicitly. Without them, the model would be unable to perform message passing and classification. Therefore, we constructed two types of node features for our GNN models.

Assume that each individual can be assigned to one of the C populations, or ancestries. If the individual's population is known, we say that the corresponding node in the graph is labeled. Otherwise, we call a node unlabeled and it is used simply to refine the graph structure.

Firstly, we construct *one-hot* features. Each node labeled with population $k \in 1, \ldots, C$, is associated with the vector

$$\mathbf{x}_i = \big(0, \ldots, 0, \underbrace{1}_{k-\text{th component}}, 0, \ldots, 0\big)^\top \in \{0,1\}^C. \tag{2}$$

Unlabeled nodes are associated with a uniform vector

$$\mathbf{x}_i = \big(\tfrac{1}{C}, \tfrac{1}{C}, \ldots, \tfrac{1}{C}\big)^\top \in [0,1]^C. \tag{3}$$

These *one-hot* features do not depend on the graph structure and, in particular, do not change if some nodes are added or removed from the graph. This property is essential for our validation protocol that is described in **Methods**.

Secondly, we introduce *graph-based* features that summarize distribution of node's neighbors according to their ancestry. Let $p(j)$ denote the ancestry label of the node j. For each node i with neighbor set N_i and edge weights $W(i,j)$, we calculate several statistics for each class $c \in \{1, 2, \ldots, C\}$

For graph-based features we used the number $n_{i,c}$ of adjacent nodes $j \in N_i$ such that $p(j) = c$, average edge weight $\bar{w}_{i,c}$, maximal edge weight $w_{i,c}^{\max}$ and the standard deviation $\sigma_{i,c}$ (see equation 4) and total number of IBD segments $\mathrm{IBD}_{i,c}$ from an individual i to population c, defined as the sum of K_{ij} over all neighbors j such that $p(j) = c$.

$$\begin{aligned} \bar{w}_{i,c} &= \operatorname*{avg}_{\substack{j \in N_i \\ p(j)=c}} W(i,j), \\ w_{i,c}^{\max} &= \max_{\substack{j \in N_i \\ p(j)=c}} W(i,j), \\ \sigma_{i,c} &= \operatorname*{std}_{\substack{j \in N_i \\ p(j)=c}} W(i,j). \end{aligned} \tag{4}$$

In this feature design, unlabeled nodes never contribute to the graph-based features of their neighbors, regardless of whether those neighbors are labeled or unlabeled. However, if an unlabeled vertex is connected to labeled vertices, those labeled neighbors do contribute to the feature vector of unlabeled node. Conversely, if a vertex has no connections to any labeled nodes, all its feature components are set to zero. Thus, only labeled vertices can influence the graph-based feature computation, ensuring that feature construction relies exclusively on known population information. Concatenating the resulting statistics across all classes yields a $5C$-dimensional vector, which forms the complete set of graph-based features as written in equation 5.

$$\mathbf{x}_i = \big(n_{i,1}, \ldots, n_{i,C}, \bar{w}_{i,1}, \ldots, \bar{w}_{i,C}, \sigma_{i,1}, \ldots, \sigma_{i,C}, w_{i,1}^{\max}, \ldots, w_{i,C}^{\max}, \mathrm{IBD}_{i,1}, \ldots, \mathrm{IBD}_{i,C}\big)^\top \tag{5}$$

Unlabeled vertices As noted earlier, some vertices in the graph may lack labels. Still, as shown in the **Results** section, it is worthwhile to retain unlabeled vertices in the graph. Such unlabeled nodes can connect to both labeled individuals and newly arrived vertices during the inferring phase, leveraging the ability of GNN models to propagate information from labeled nodes through unlabeled nodes via the message-passing framework, thereby improving accuracy. This idea resonates with broader evidence from unsupervised and self-supervised learning. In computer vision, clustering methods such as SCAN have shown that leveraging unlabeled samples through nearest-neighbor consistency can enhance semantic feature learning and improve downstream classification accuracy (Van Gansbeke et al., 2020). A related trend is observed in industrial process monitoring, where SensorSCAN demonstrates that self-supervised pretraining on unlabeled sensor data, followed by clustering, yields robust latent representations that boost fault detection even with very limited labels (Golyadkin et al., 2023).

3 METHOD

We pose ancestry inference as a node classification task in which each vertex represents an individual and edges reflect genetic similarity between them. The goal is to predict a most probable population

label of the nodes, where label of each node belongs to one of C classes. We evaluate graph neural networks that operate on node features and edge weights, and we compare them with four non-GNN baselines.

The first baseline is the multiclass logistic regression (LR) on the vector of node features 5 (see details in Appendix C). The second baseline includes graph community detection algorithms that use only the graph structure and known node classes without constructing node features (see details in Appendix D). The third baseline is a multilayer perceptron (MLP) that relies solely on graph based node features and ignores the graph structure.

Our practical setting requires generalization to new vertices that arrive over time, which makes purely transductive methods Kipf & Welling (2016); Huang et al. (2020); Oono & Suzuki (2020) inappropriate since they assume a fixed graph during both training and inference. We therefore adopt a semi inductive strategy in which we train on a fixed graph topology while deliberately augmenting node features to enable generalization to graphs with different structure at validation and test time.

Reference graph and dataset split In our framework, new individuals arrive incrementally, and for each of them we compute genetic similarities to all existing nodes in the database. To manage this process consistently, we rely on a structure called the *reference graph*, which serves as the graph used for training and as the fixed knowledge base for inductive validation and testing. It contains all vertices available during training, including both labeled and unlabeled nodes (if present).

For our experiments each dataset is split into training, validation, and test nodes in proportions of 60%, 20%, and 20%, respectively. We generate ten independent random splits to ensure robust estimates. The training nodes constitute the reference graph, which may already include some isolated nodes. Validation and test graphs are then constructed by independently adding one validation or test node at a time to the reference graph. For each node in the validation or test set (inference stage), we add exactly one validation or test node to the reference graph consisting of all the training nodes (see cases A or B in Supplementary Figure 2). If no edges exist between the newly added node and the reference graph, the prediction for that individual cannot be performed and the pipeline skips this node. Such isolated validation or test vertices do not contribute to any evaluation metrics reported in this work.

The reference graph remains fixed with its node and edge features unchanged at the inference stage. This setup ensures that each prediction is performed independently and that new nodes remain invisible to one another during classification. Moreover, because only nodes with known ancestry contribute to the features of their neighbors, the inference target node does not modify any existing node features. Both one-hot and graph-based feature representations satisfy this requirement, making them suitable for inductive inference. In contrast, embedding-based approaches such as node2vec Grover & Leskovec (2016) are unsuitable for this setting, since even small topological changes can produce entirely different node embeddings.

Models and training regimes We compare four classes of methods under a unified experimental framework. Graph neural networks (GNNs) exploit the complete graph structure, utilizing both node features and edge weights while performing message passing across all available connections. Multi layer perceptrons (MLPs) and logistic regression (LR) models rely exclusively on precomputed graph based node features that summarize local connectivity but do not incorporate explicit graph topology. In contrast, classical community detection algorithms do not learn parameters and are evaluated only at test time on the reference graph together with one added test node at a time, following the inductive validation protocol described earlier.

For GNNs, we use two types of node features, one hot and graph based, both trained under the same semi inductive regime (see Appendix E). During training, the label is removed for a randomly chosen labeled node at each step. For one hot features, this means replacing the one hot vector with a uniform vector as defined in equation 3. For graph based features, the procedure is slightly trickier to avoid data leak: the features of the chosen node's neighbors are updated by removing the contribution of this node into the feature vector equation 5.

The selected vertex is treated as unlabeled for the duration of that iteration by recalculating the features of its neighbors so that this vertex no longer contributes to their class dependent statistics.

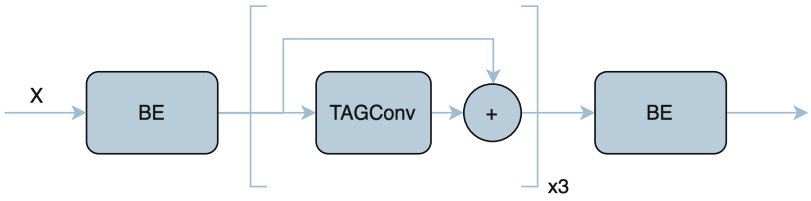

Figure 1: Best architecture with 3 TAGConv layers and Batch Ensemble projectors

The graph based features of the vertex itself remain unchanged, since the class composition among its neighbors does not change when the vertex is considered unlabeled.

GNN models utilize unlabeled vertices whenever they are available, unless explicitly stated otherwise, as these nodes can contribute to message passing and improve learning through their structural connections. In both node feature cases, during training the loss is computed only for the vertex currently treated as unlabeled using the weighted cross entropy, with class probabilities determined by the softmax and class weights which normalizes class counts. This procedure ensures that the training process closely matches the inference stage, where a single new node is introduced and classified while the rest of the reference graph remains unchanged.

MLP and LR models are trained in a standard fully supervised manner using fixed graph based node features. In this case, the loss is computed for all labeled vertices and then averaged. Unlabeled vertices are excluded from training because their true population affiliations are unknown.

Improved GNN architecture Our base GNN models consist of a stack of graph convolutional layers (see Supplementary Figure 3). While such architectures effectively incorporate graph structure, they are known to suffer from sensitivity to parameter initialization (Klabunde & Lemmerich, 2022) and from node representation oversmoothing (Rusch et al., 2023b). To improve robustness and representation quality, we introduce two complementary architectural modifications that target these issues.

Repeated graph convolutions tend to progressively oversmooth node representations, causing the model to lose informative low-level feature signals. To address this issue, we incorporate skip-connections between graph convolution layers. These connections allow the network to reuse low-level feature representations on each layer, while still using structural information. Additionally, we add linear projectors before the first and after the last graph convolution layers to map hidden vectors to a common dimensionality, which enables the skip-connections to be applied consistently.

While skip-connections mitigate oversmoothing, they do not address initialization dependence in GNN training. To improve robustness to initialization, we draw inspiration from tabular prediction. If we ignore the graph structure and consider ancestry prediction based only on graph-based features, the task becomes closely related to tabular deep learning, where each sample is represented as an independent feature vector. This perspective motivates borrowing architectural ideas from modern tabular models, in particular the TabM architecture (Gorishniy et al., 2025), which replaces standard MLP layers with BatchEnsemble layers and improves performance on a wide range of tabular tasks.

In a BatchEnsemble linear layer, each ensemble member is parameterized by lightweight adapters R_i, S_i, B_i and a shared weight matrix W. The transformation is defined as

$$((X \circ R_i)W) \circ S_i + B_i,$$

where $\circ$ denotes element-wise multiplication. To reduce sensitivity to initialization and improve the diversity of learned representations, we replace the linear projection layers in our architecture with BatchEnsemble linear layers. The resulting model, which we call `GNNM` (**GNN** that makes **M**ultiple predictions), first projects the input features using a BatchEnsemble layer, then processes them through multiple graph convolution blocks with skip-connections, and finally produces logits using a final BatchEnsemble layer. The full architecture is illustrated in Figure 1.

4 RESULTS

We consider five datasets in total, each represented as a single graph without isolated vertices and containing different populations grouped by regions of the world. Our primary analysis focuses on the Eastern Slavs (ES) dataset, which includes both labeled and unlabeled vertices, reflects closely related population groups, and most accurately represents real population structure. Unlike other datasets, ES underwent minimal preprocessing and is the only one that naturally contains real unlabeled individuals. To demonstrate the robustness and versatility of our approach, we additionally experiment with four supplementary datasets containing different labeled populations, with full details provided in Appendix M. Those datasets contain only labeled vertices with known population affiliations and therefore do not include real unlabeled nodes.

ES labeled-only experiments We evaluate three model families — GNNs, MLPs, LR, as well as community detection algorithms — on the Eastern Slavs (ES) dataset. For this dataset, we first identify the best-performing GNN architecture by comparing 29 different model configurations with varying depths, widths, and convolutional layers (see Appendix F & G). Specifically, we experiment with several widely used graph convolution operators implemented in PyTorch Geometric framework, including TAGConv (Du et al., 2017), GATConv (Veličković et al., 2017), GINConv (Xu et al., 2018), GraphSAGE (Hamilton et al., 2017), and GCNConv (Kipf & Welling, 2016). The model `GL_TAGConv_3l_512h_w_k3_gnorm` consistently demonstrates the best performance in both variants, trained with one-hot features and with graph-based features. The strongest MLP model was `GL_MLP_3l_512h`.

On the labeled-only ES graph, the one-hot GNN reaches 0.6165 ± 0.02 f1-macro and the graph-based GNN achieves 0.6146 ± 0.02. The best MLP and LR reach 0.6125 ± 0.02 and 0.6059 ± 0.02, while community detection performs lower at 0.5307 ± 0.01. These values fall within similar standard deviations, indicating that the small gaps observed on the fully labeled graph are not statistically significant. Results on other datasets are available in Appendix H, where we also show that community detection methods consistently achieved the lowest accuracy across all datasets. Therefore, we report them only in the labeled-only setting and exclude them from all subsequent analyses in this paper.

Edge weights importance To quantify the effect of incorporating edge weights representing total IBD segment length, we performed an experiment using one of the best-performing architectures, `GL_TAGConv_3l_512h_w_k3_gnorm` (with edge weights) and `GL_TAGConv_3l_512h_nw_k3_gnorm` (same architecture but without edge weights). Both models were trained with one hot node features only. The evaluation was carried out on five fully labeled real graphs — SC, WE, NC, VU, and ES. The ES dataset used here was also fully labeled and contained no unlabeled vertices. The topology of the graphs and the node features were identical in both weighted and unweighted configurations. The only difference was that one GNN architecture utilized edge weights while the other ignored them. Identical data splits, hyperparameters, and optimization settings were applied across both versions of each dataset to ensure that any difference in performance reflects only the contribution of the edge-weight attribute.

The results summarized in Table 1 show that including edge weights slightly improves model performance on every dataset, though the gains remain small and typically fall within one standard deviation across splits. On average, models with edge weights achieve marginally higher f1-macro scores. This suggests that the total IBD segment length contributes a weak yet consistently informative weighting signal for genetic similarity. Based on this evidence, we retain edge weights in all subsequent experiments reported in the paper.

Unlabeled experiments After establishing these baselines, we examine the role of unlabeled data using artificially generated unlabeled vertices (see Appendix I.1). For each dataset that does not naturally contain unlabeled nodes, including the version of ES where all vertices are labeled, we randomly mark a subset of labeled vertices as intentionally unlabeled to simulate partially observed population structure. The same training, validation, and testing pipeline used later for ES with real unlabeled individuals is applied here to maintain methodological consistency. By varying the proportion of artificially unlabeled vertices relative to the remaining labeled ones, we observe a clear and consistent trend — the addition of unlabeled vertices improves the accuracy of GNN

Table 1: Results of training GNN models on fully labeled graphs with and without edge weights. Values are means and standard deviations over 10 splits.

Dataset	With weights (w)		Without weights (nw)	
	Mean	Std	Mean	Std
SC	0.9781	0.01	0.9766	0.01
WE	0.9462	0.01	0.9432	0.01
NC	0.9645	0.02	0.9313	0.03
ES	0.6114	0.02	0.5917	0.02
VU	0.9971	0.01	0.9912	0.01

Table 2: ES baselines and extended results with real unlabeled nodes. The GNN architecture used for both feature variants is `GL_TAGConv_3l_512h_w_k3_gnorm`. MLP (`GL_MLP_3l_512h` was the most accurate here), LR and community detection algorithms (best is `LabelPropagation`) are not applicable in the unlabeled setting.

% unlabeled nodes	GNN (graph-based)	GNN (one-hot)	MLP	LR	Community Detection
0%	0.6146 ± 0.02	0.6165 ± 0.02	0.6125 ± 0.02	0.6059 ± 0.02	0.5307 ± 0.01
1%	0.6285 ± 0.01	0.6255 ± 0.01	NA	NA	NA
5%	0.6500 ± 0.02	0.6467 ± 0.02	NA	NA	NA
25%	0.6844 ± 0.02	0.6672 ± 0.01	NA	NA	NA
50%	0.6962 ± 0.02	0.6765 ± 0.01	NA	NA	NA
100%	$\mathbf{0.7135 \pm 0.01}$	0.6825 ± 0.01	NA	NA	NA

models across nearly all datasets. This confirms that the positive effect of incorporating unlabeled individuals is general and not tied to any specific regional population.

Importantly, the artificial masking experiments also show that correct predictions remain possible even when a validation or test vertex is connected only to unlabeled neighbors. As detailed in Appendix I.1, we explicitly analyze such cases and observe that message passing through unlabeled intermediates can still convey sufficient structural information for accurate classification.

Then we tested our best performing model `GL_TAGConv_3l_512h_w_k3_gnorm` on extended ES dataset with unknown labels with both one-hot and graph-based features. Let $\rho \in \{0, 1, 5, 25, 50, 100\}\%$ denote the fraction of real unlabeled vertices that are added to the labeled ES graph. For each $\rho > 0$, we randomly select $\rho\%$ of the available unlabeled vertices and merge them with the labeled ES graph, evaluating all models using 10 random train/validation/test splits as described earlier. Baseline models (MLP, LR, and community detection) are measured once on the labeled-only ES graph ($\rho = 0\%$) and shown in Table 2.

When real unlabeled vertices are gradually added to the graph ($\rho > 0$), both GNN variants show steady improvement. The one hot GNN improves from about 0.62 to 0.68 f1-macro as ρ grows from 0% to 100%, whereas the graph based GNN rises from approximately 0.61 to 0.71, consistently outperforming the one hot configuration for all $\rho > 1\%$. The improvement becomes more pronounced as unlabeled vertices constitute a larger portion of the graph, confirming that unlabeled vertices provide additional relational context that can be exploited through message passing.

Overall, the results highlight that the inclusion of real unlabeled individuals in the ES dataset and the use of message passing through unlabeled intermediates substantially enhance classification accuracy in realistic semi-inductive settings. The validation and test parts of the ES dataset contain only one vertex whose neighborhood consists exclusively of unlabeled vertices.

The results demonstrate a substantial improvement in classification accuracy once unlabeled individuals are introduced, showing that unlabeled nodes provide valuable structural information that enhances the model's ability to identify population groups correctly. We benchmark only GNNs in this setting because, as explained in Appendix J, MLP, LR, and community detection approaches cannot effectively operate when big amount of unlabeled vertices are present in the graph.

Table 3: Final performance (same f1 macro score across all populations) of different GNN architectures using 100% unlabeled nodes with different input feature types. Baseline here is GL_TAGConv_3l_512h_w_k3_gnorm.

Input features	Baseline	GL_TAGConv_3l_512h_w_k3_gnorm_sk	GL_GNNM
One-hot	0.6825 ± 0.01	0.7205 ± 0.02	0.7231 ± 0.02
Graph-based	0.7135 ± 0.01	0.7251 ± 0.01	$\mathbf{0.7346 \pm 0.02}$

Simulated data experiments Next, we explore the robustness of our models using both real and simulated graphs with different levels of sparsity I.2. These experiments are conducted on the VU dataset, which is the densest graph in our collection and therefore suitable for systematically reducing connectivity across a broad range of levels. As edges are progressively removed, all models experience some degree of performance degradation, which is expected as the structural information in the graph becomes less informative. However, GNNs consistently remain slightly more robust than other approaches, preserving higher accuracy under moderate and even severe sparsity.

Then, we assess how class imbalance affects the overall performance of our models. As described earlier, all real datasets contain uneven distributions of individuals across population groups. To isolate this factor, we use simulated ES graphs without unlabeled nodes and vary class proportions while keeping the total number of vertices constant. The results reveal that class imbalance has a negligible effect on model accuracy. GNNs and MLPs remain largely stable due to the use of class-weighted loss functions. Detailed results of the class imbalance experiments are provided in the Supplementary Material (see Section I.3).

Model enhancements We selected GL_TAGConv_3l_512h_w_k3_gnorm as our best-performing baseline GNN configuration and modified it with two different variants. In the first modification, we added skip-connections between TAGConv layers (GL_TAGConv_3l_512h_w_k3_gnorm_sk), allowing the model to use low-level features throughout the network. In the second variant (GL_GNNM), we kept the skip-connections and replaced the linear projection layers with BatchEnsemble layers initialized following the TabM approach (Gorishniy et al., 2025).

We evaluated these three models on the ES dataset, using all unlabeled nodes, with both one hot and graph-based features. Each model was trained and evaluated across 10 data splits as in all previous experiments. As shown in Table 3, both proposed modifications consistently improved predictive performance in both the one hot and graph-based feature settings.

5 Conclusion

In this work, we introduced a graph-based approach called GENLINK for ancestry inference from identity-by-descent (IBD) data, where each individual is represented as a node and edges encode the amount of shared genetic segments. Our method employs Graph Neural Networks (GNNs) to propagate known population labels through message passing and assign ancestry labels in an inductive setting. The framework relies on simple and interpretable node features, including one-hot encodings and graph-based topological statistics, and avoids retraining when the graph evolves during inference. Across multiple population-genetic datasets, the proposed GNN models consistently outperformed all baseline algorithms.

We further demonstrated that incorporating unlabeled individuals improves predictive performance by enriching the graph topology and enabling more effective propagation of ancestry information. Edge weighting by total IBD length provided only marginal gains, indicating that structural connectivity plays a dominant role. To enhance robustness, we introduced architectural improvements including skip-connections to preserve low-level feature representations and BatchEnsemble-based linear projections to mitigate training instability. Extensive ablation studies on edge sparsity, class imbalance, and within-class connectivity confirmed that GNN-based approaches remain resilient under substantial graph degradation. On the Eastern Slavs dataset, which contains numerous unlabeled individuals and closely related populations, the GNN with graph-based features achieved the

strongest performance, highlighting the effectiveness of message passing in partially labeled genetic graphs.

Future work will extend this framework beyond single-label classification toward estimating admixture proportions, that is continuous ancestry fractions from multiple populations. Such an extension represents a more complex yet practically important objective. We anticipate that a graph-based learning strategy adapted to produce multi-ancestry profiles can further improve the resolution of ancestry inference while preserving the efficiency and inductive capabilities demonstrated in this study. Runtime and memory statistics are provided in Appendix K.

REFERENCES

Genetic risk and a primary role for cell-mediated immune mechanisms in multiple sclerosis. *Nature*, 476(7359):214–219, 2011.

David H Alexander, John Novembre, and Kenneth Lange. Fast model-based estimation of ancestry in unrelated individuals. *Genome research*, 19(9):1655–1664, 2009.

Doron M Behar, Bayazit Yunusbayev, Mait Metspalu, Ene Metspalu, Saharon Rosset, Jüri Parik, Siiri Rootsi, Gyaneshwer Chaubey, Ildus Kutuev, Guennady Yudkovsky, et al. The genome-wide structure of the jewish people. *Nature*, 466(7303):238–242, 2010.

Doron M Behar, Mait Metspalu, Yael Baran, Naama M Kopelman, Bayazit Yunusbayen, Ariella Gladstein, Shay Tzur, Hovhannes Sahakyan, Ardeshir Bahmanimehr, Levon Yepiskoposyan, et al. No evidence from genome-wide data of a khazar origin for the ashkenazi jews. *Human biology*, 85(6):859–900, 2013.

Brian L Browning and Sharon R Browning. Improving the accuracy and efficiency of identity-by-descent detection in population data. *Genetics*, 194(2):459–471, 2013.

Brian L Browning, Ying Zhou, and Sharon R Browning. A one-penny imputed genome from next-generation reference panels. *The American Journal of Human Genetics*, 103(3):338–348, 2018.

Howard M Cann, Claudia De Toma, Lucien Cazes, Marie-Fernande Legrand, Valerie Morel, Laurence Piouffre, Julia Bodmer, Walter F Bodmer, Batsheva Bonne-Tamir, Anne Cambon-Thomsen, et al. A human genome diversity cell line panel. *Science*, 296(5566):261–262, 2002.

1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68, 2015.

Jian Du, Shanghang Zhang, Guanhang Wu, José MF Moura, and Soummya Kar. Topology adaptive graph convolutional networks. *arXiv preprint arXiv:1710.10370*, 2017.

William A Freyman, Kimberly F McManus, Suyash S Shringarpure, Ethan M Jewett, Katarzyna Bryc, 23, Me Research Team, and Adam Auton. Fast and robust identity-by-descent inference with the templated positional burrows–wheeler transform. *Molecular Biology and Evolution*, 38 (5):2131–2151, 2021.

Maksim Golyadkin, Vitaliy Pozdnyakov, Leonid Zhukov, and Ilya Makarov. Sensorscan: Self-supervised learning and deep clustering for fault diagnosis in chemical processes. *Artificial Intelligence*, 324:104012, 2023.

Yury Gorishniy, Akim Kotelnikov, and Artem Babenko. Tabm: Advancing tabular deep learning with parameter-efficient ensembling, 2025. URL https://arxiv.org/abs/2410.24210.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R Benson. Combining label propagation and simple models out-performs graph neural networks. *arXiv preprint arXiv:2010.13993*, 2020.

Choongwon Jeong, Oleg Balanovsky, Elena Lukianova, Nurzhibek Kahbatkyzy, Pavel Flegontov, Valery Zaporozhchenko, Alexander Immel, Chuan-Chao Wang, Olzhas Ixan, Elmira Khussainova, et al. The genetic history of admixture across inner eurasia. *Nature ecology & evolution*, 3 (6):966–976, 2019.

Tomasz Kajdanowicz. Relational classification using random walks in graphs. *New Generation Computing*, 33:409–424, 2015.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Max Klabunde and Florian Lemmerich. On the prediction instability of graph neural networks, 2022. URL https://arxiv.org/abs/2205.10070.

PS Kostenetskiy, RA Chulkevich, and VI Kozyrev. Hpc resources of the higher school of economics. In *Journal of Physics: Conference Series*, volume 1740, pp. 012050. IOP Publishing, 2021.

Iosif Lazaridis, Nick Patterson, Alissa Mittnik, Gabriel Renaud, Swapan Mallick, Karola Kirsanow, Peter H Sudmant, Joshua G Schraiber, Sergi Castellano, Mark Lipson, et al. Ancient human genomes suggest three ancestral populations for present-day europeans. *Nature*, 513(7518):409–413, 2014.

Yu Li, Siyuan Chen, Trisevgeni Rapakoulia, Hiroyuki Kuwahara, Kevin Y Yip, and Xin Gao. Deep learning identifies and quantifies recombination hotspot determinants. *Bioinformatics*, 38(10): 2683–2691, 2022.

Miriam F Moffatt, Ivo G Gut, Florence Demenais, David P Strachan, Emmanuelle Bouzigon, Simon Heath, Erika von Mutius, Martin Farrall, Mark Lathrop, and William OCM Cookson. A large-scale, consortium-based genomewide association study of asthma. *New England Journal of Medicine*, 363(13):1211–1221, 2010.

Kenta Oono and Taiji Suzuki. Optimization and generalization analysis of transduction through gradient boosting and application to multi-scale graph neural networks. *Advances in Neural Information Processing Systems*, 33:18917–18930, 2020.

Maanasa Raghavan, Pontus Skoglund, Kelly E Graf, Mait Metspalu, Anders Albrechtsen, Ida Moltke, Simon Rasmussen, Thomas W Stafford Jr, Ludovic Orlando, Ene Metspalu, et al. Upper palaeolithic siberian genome reveals dual ancestry of native americans. *Nature*, 505(7481):87–91, 2014.

T Konstantin Rusch, Michael M Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023a.

T. Konstantin Rusch, Michael M. Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks, 2023b. URL https://arxiv.org/abs/2303.10993.

Sahotra Sarkar. Anecdotal, historical and critical commentaries on genetics. *Genetics*, 142(3): 655–660, 1996.

Shi. Multiclass spectral clustering. In *Proceedings ninth IEEE international conference on computer vision*, pp. 313–319. IEEE, 2003.

Kristiina Tambets, Bayazit Yunusbayev, Georgi Hudjashov, Anne-Mai Ilumäe, Siiri Rootsi, Terhi Honkola, Outi Vesakoski, Quentin Atkinson, Pontus Skoglund, Alena Kushniarevich, et al. Genes reveal traces of common recent demographic history for most of the uralic-speaking populations. *Genome biology*, 19(1):139, 2018.

Petr Triska, Nikolay Chekanov, Vadim Stepanov, Elza K Khusnutdinova, Ganesh Prasad Arun Kumar, Vita Akhmetova, Konstantin Babalyan, Eugenia Boulygina, Vladimir Kharkov, Marina Gubina, et al. Between lake baikal and the baltic sea: genomic history of the gateway to europe. *BMC genetics*, 18(Suppl 1):110, 2017.

Wouter Van Gansbeke, Simon Vandenhende, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool. Scan: Learning to classify images without labels. In *European conference on computer vision*, pp. 268–285. Springer, 2020.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Peter Vojtek and Mária Bieliková. Homophily of neighborhood in graph relational classifier. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pp. 721–730. Springer, 2010.

Xinyi Wu, Zhengdao Chen, William Wang, and Ali Jadbabaie. A non-asymptotic analysis of oversmoothing in graph neural networks. *arXiv preprint arXiv:2212.10701*, 2022.

Xinyi Wu, Amir Ajorlou, Zihui Wu, and Ali Jadbabaie. Demystifying oversmoothing in attention-based graph neural networks. *Advances in Neural Information Processing Systems*, 36:35084–35106, 2023.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Bayazit Yunusbayev, Mait Metspalu, Mari Järve, Ildus Kutuev, Siiri Rootsi, Ene Metspalu, Doron M Behar, Kärt Varendi, Hovhannes Sahakyan, Rita Khusainova, et al. The caucasus as an asymmetric semipermeable barrier to ancient human migrations. *Molecular biology and evolution*, 29(1): 359–365, 2012.

Bayazit Yunusbayev, Mait Metspalu, Ene Metspalu, Albert Valeev, Sergei Litvinov, Ruslan Valiev, Vita Akhmetova, Elena Balanovska, Oleg Balanovsky, Shahlo Turdikulova, et al. The genetic legacy of the expansion of turkic-speaking nomads across eurasia. *PLoS genetics*, 11(4): e1005068, 2015.

Zan Zhang, Hao Wang, Lin Liu, and Jiuyong Li. Multi-label relational classification via node and label correlation. *Neurocomputing*, 292:72–81, 2018.

Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. *ProQuest number: information to all users*, 2002.

A Dataset Preparation

Genetic data preparation and datasets We analyzed genetic data from 7200 individuals drawn from the Genotek ancestry reference panel `www.genotek.ru`. The panel aggregates previously published genomic resources that represent diverse populations (Tables 4, 5, 6 and 7) together with data from Genotek clients who self reported the ancestry of their grandparents. We organized all individuals into five region specific datasets used in this study, Northern Caucasus dataset (NC), Western Europe dataset (WE), Volga Ural region dataset (VU), Scandinavia dataset (SC), and Eastern Slavs dataset (ES). The ES dataset is fully proprietary, it is provided by Genotek, and it is not publicly available. All preparation steps described below were applied to each of the five datasets.

All participating Genotek clients provided informed consent for the use of their genetic information for research and completed an online questionnaire. The present study was approved by the Genotek Ethics Committee under protocol No. 14, Graph Neural Network application for accurate ancestry inference from identity by descent graphs, and was conducted in accordance with the Declaration of Helsinki.

For the Genotek client cohort we extracted DNA from saliva and performed genotyping on Illumina Infinium Global Screening Array versions 1 through 3 microarrays that assay approximately 650000 single nucleotide polymorphisms (SNPs). Samples were processed in discrete batches of 192 to 768 individuals. Genotype calling and clustering of raw signals were carried out in GenomeStudio software, Illumina, San Diego, CA, with manually created cluster files. We removed SNPs with within batch call rate below 0.9 and we excluded individuals whose sample call rate was below 0.97.

Previously published cohorts were genotyped on different microarrays. To harmonize all sources we imputed genotypes with the Haplotype Reference Consortium and the 1000 Genomes reference panels using Beagle 5.1 (Browning et al., 2018). Only variants with imputation quality DR2 above 0.7 were retained for downstream analyses.

Dataset description We inferred identity by descent segments using the TPBWT implementation available in the Python package `phasedibd` (Freyman et al., 2021). We set the minimal cumulative segment length threshold to 6 centimorgans and additionally required at least one shared IBD segment of length not less than 8 centimorgans. A centimorgan is a unit of genetic linkage defined as a one percent probability of recombination per generation. Two markers separated by 1 centimorgan therefore have a one percent chance to be separated by crossing over during gamete formation (Sarkar, 1996). Although a centimorgan measures genetic distance rather than physical length in humans it typically corresponds to roughly one million base pairs and this correspondence varies across genomic regions and between species. These inferred segments serve as the basis for the similarity measure and the graph representation used in subsequent analyses.

For NC, WE, VU, and SC, provenance for nodes linked to published cohorts is summarized in Tables 4, 5, 6 and 7. For each dataset, the total sum of node counts n equals the total number of unique nodes in that dataset.

Publication	n
A human genome diversity cell line panel (Cann et al., 2002)	17
The genome wide structure of the Jewish people (Behar et al., 2010)	3
The Caucasus as an Asymmetric Semipermeable Barrier to Ancient Human Migrations (Yunusbayev et al., 2012)	48
No evidence from genome wide data of a Khazar origin for the Ashkenazi Jews (Behar et al., 2013)	4
Ancient human genomes suggest three ancestral populations for present day Europeans (Lazaridis et al., 2014)	20
The Genetic Legacy of the Expansion of Turkic Speaking Nomads across Eurasia (Yunusbayev et al., 2015)	6
Between Lake Baikal and the Baltic Sea, genomic history of the gateway to Europe (Triska et al., 2017)	125
The genetic history of admixture across inner Eurasia (Jeong et al., 2019)	104
Genotek proprietary data	287

Table 4: Publications contributing samples to the Northern Caucasus dataset (NC).

Publication	n
A human genome diversity cell line panel (Cann et al., 2002)	4
A large scale, consortium based genomewide association study of asthma (Moffatt et al., 2010)	499
A large scale, consortium based genomewide association study of asthma (Moffatt et al., 2010)	1511
Genetic risk and a primary role for cell mediated immune mechanisms in multiple sclerosis (int, 2011)	1796
A global reference for human genetic variation (Consortium et al., 2015)	47
Genotek proprietary data	0

Table 5: Publications contributing samples to the Western Europe dataset (WE).

Publication	n
A large scale, consortium based genomewide association study of asthma (Moffatt et al., 2010)	198
Upper Palaeolithic Siberian genome reveals dual ancestry of Native Americans (Raghavan et al., 2014)	12
Ancient human genomes suggest three ancestral populations for present day Europeans (Lazaridis et al., 2014)	2
The Genetic Legacy of the Expansion of Turkic Speaking Nomads across Eurasia (Yunusbayev et al., 2015)	30
Between Lake Baikal and the Baltic Sea, genomic history of the gateway to Europe (Triska et al., 2017)	46
Genes reveal traces of common recent demographic history for most of the Uralic speaking populations (Tambets et al., 2018)	2
The genetic history of admixture across inner Eurasia (Jeong et al., 2019)	40
Genotek proprietary data	1312

Table 6: Publications contributing samples to the Volga Ural region dataset (VU).

Publication	n
A large scale, consortium based genomewide association study of asthma (Moffatt et al., 2010)	108
Genetic risk and a primary role for cell mediated immune mechanisms in multiple sclerosis (int, 2011)	971
No evidence from genome wide data of a Khazar origin for the Ashkenazi Jews (Behar et al., 2013)	1
Ancient human genomes suggest three ancestral populations for present day Europeans (Lazaridis et al., 2014)	6
Genotek proprietary data	0

Table 7: Publications contributing samples to the Scandinavia dataset (SC).

B EASTERN SLAVS DATA SOURCE

The Eastern Slavs (ES) dataset consists of four population classes representing individuals from distinct geographical areas of Eastern Europe. Each class corresponds to a specific set of regions and cities that characterize its population distribution. Below we list the composition of each class.

Northern Russians. This group includes individuals from Nizhny Novgorod, Ivanovo, Yaroslavl, Vologda, Arkhangelsk, Kostroma, Novgorod, Tver, and Murmansk regions, as well as from the Republic of Karelia. No specific cities were explicitly listed for this group.

Southern Russians. This class represents individuals from Kursk, Belgorod, Penza, Tambov, Oryol, Bryansk, and Voronezh regions. The corresponding cities mentioned include Oryol, Penza, and Michurinsk.

Ukrainians. This group includes individuals from Poltava, Lviv, Zakarpattia, Sumy, Dnipropetrovsk, Vinnytsia, Kyiv, Kirovohrad, Odesa, Zhytomyr, Cherkasy, Kharkiv, Ivano-Frankivsk, Chernihiv, Chernivtsi, Volyn, Ternopil, Mykolaiv, Khmelnytskyi, and Rivne regions. The listed cities are Kyiv and Delyatyn.

Belarusians. This class contains individuals from Mogilev, Minsk, Gomel, Vitebsk, Brest, and Grodno regions. The associated cities are Minsk and Baranovichi.

C MULTICLASS LOGISTIC REGRESSION

We use a multinomial logistic regression classifier that operates on the *graph based* node features defined in 5, which summarize structural statistics of the labeled neighborhood but do not rely on the graph topology during training. For each node i with feature vector $\mathbf{x}_i$, class scores are computed as

$$\mathbf{z}_i = W\,\mathbf{x}_i + \mathbf{b}, \tag{6}$$

and converted into class probabilities using the softmax function

$$p_{i,c} = \frac{\exp(z_{i,c})}{\sum_{k=1}^{C} \exp(z_{i,k})}. \tag{7}$$

The model is trained by minimizing the regularized multinomial negative log-likelihood over labeled nodes

$$\mathcal{L}_{\mathrm{LR}} = -\sum_{i \in \mathcal{V}_{\mathrm{train}}} \log p_{i,y_i} + \lambda \|W\|_2^2, \tag{8}$$

where λ controls the L2 regularization strength. In our implementation, we use the default setting of the `scikit-learn` `LogisticRegression` class with `C`=1.0, corresponding to $\lambda = 1.0$.

Before training, all feature vectors are standardized using `StandardScaler` to achieve zero mean and unit variance across dimensions. The model is implemented through an `sklearn` pipeline that performs feature normalization and fits a multinomial logistic regression model with the `lbfgs` solver and `max_iter`$= 500$:

- `make_pipeline(StandardScaler(), LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=500))`

The model runs entirely on CPU and makes predictions solely from fixed *graph based* features without performing message passing or using edge information.

D COMMUNITY DETECTION ALGORITHMS

We employed several community detection algorithms to classify nodes in our graph, serving as comparative baselines for our Graph Neural Network (GNN) models. These algorithms utilize the graph structure and connectivity patterns to infer node labels based on the assumption that connected nodes are likely to belong to the same class. Below, we describe each algorithm in detail.

Label Propagation Label Propagation Zhu & Ghahramani (2002) is an iterative algorithm that infers the labels of unlabeled nodes by propagating label information through the graph structure. The algorithm operates by updating the label distributions based on the labels of neighboring nodes, weighted by the graph's adjacency relationships.

Formally, let $G = (V, E)$ be an undirected graph with nodes V and edges E. Let A be the adjacency matrix of G, and D be the degree matrix where $D_{ii} = \sum_j A_{ij}$. Let Y be a matrix where each row corresponds to a node's label distribution, with known labels encoded as one-hot vectors and unknown labels initialized as zero vectors.

The label propagation operator is defined as:

$$Y' = \alpha \cdot D^{-1/2} A D^{-1/2} Y + (1 - \alpha) Y,$$

where $\alpha \in (0, 1)$ is a hyperparameter controlling the extent of propagation versus retention of initial labels.

The algorithm proceeds iteratively as follows:

1. Initialization: For labeled nodes v, the label distributions Y_v are set to their one-hot encoded labels. For unlabeled nodes v, the label distributions Y_v are initialized to zero vectors.

2. Iteration: At each iteration t, update the label distributions:

$$Y^{(t+1)} = \alpha \cdot D^{-1/2} A D^{-1/2} Y^{(t)} + (1 - \alpha) Y^{(0)},$$

where $Y^{(0)}$ is the initial label distribution matrix.

3. Normalization: After each update, normalize the label distributions so that each node's label distribution sums to one.

4. Convergence: Repeat the iteration until convergence, i.e., until $Y^{(t+1)}$ is sufficiently close to $Y^{(t)}$.

In this formulation, the term $D^{-1/2} A D^{-1/2}$ represents the normalized adjacency matrix, which accounts for the degree of each node, ensuring that label propagation is properly scaled across the graph. The parameter α controls the influence of neighboring nodes' labels versus the retention of a node's initial label distribution.

In our implementation, we use the `LabelPropagation` model Huang et al. (2020) from the PyTorch Geometric library, which follows this formulation.

Spectral Clustering Spectral Clustering Shi (2003) leverages the eigenvalues and eigenvectors of the graph Laplacian to perform clustering. The key idea is to project the data into a lower-dimensional space using the eigenvectors corresponding to the smallest eigenvalues, where the graph's connectivity is preserved.

Given the graph $G = (V, E)$ with adjacency matrix A, the unnormalized graph Laplacian is defined as:

$$L = D - A$$

where D is the degree matrix, a diagonal matrix with $D_{ii} = \sum_j A_{ij}$.

Alternatively, the normalized Laplacian is:

$$L_{\text{sym}} = D^{-1/2}LD^{-1/2} = I - D^{-1/2}AD^{-1/2}$$

The Spectral Clustering algorithm involves the following steps:

1. Compute the Laplacian matrix L or L_{sym}.

2. Compute the first k eigenvectors $u_1, u_2, \ldots, u_k$ corresponding to the smallest eigenvalues of L.

3. Form a matrix $U \in \mathbb{R}^{n \times k}$ where each row corresponds to a node and columns are the components of the eigenvectors.

4. Normalize the rows of U to have unit length.

5. Apply k-means clustering to the rows of U, assigning nodes to clusters.

In our method, we set k equal to the number of classes and use the 'lobpcg' eigen_solver for computational efficiency from Python scikit-learn library. We apply the clustering to the subgraph containing the test node and its connected training nodes, ensuring the test node is assigned to one of the clusters. We then map clusters to classes by matching cluster labels to the most frequent class label within each cluster.

Agglomerative Clustering Agglomerative Clustering is a hierarchical clustering method that builds clusters by recursively merging the pair of clusters that minimally increases a given linkage distance. In our context, we use the complete linkage criterion, which considers the maximum distance between elements of each cluster.

The algorithm proceeds as follows:

1. Initialize each node as its own cluster.

2. Compute the pairwise distance matrix between clusters. In our case, we use the SimRank distance, which measures the similarity between nodes based on the idea that two nodes are similar if they are connected to similar nodes.

The SimRank similarity $s(a, b)$ between nodes a and b is defined recursively:

$$s(a,b) = \begin{cases} 1 & \text{if } a = b \\ \frac{C}{|N(a)| \cdot |N(b)|} \sum_{i=1}^{|N(a)|} \sum_{j=1}^{|N(b)|} s(N_i(a), N_j(b)) & \text{if } a \neq b \end{cases}$$

where $N(a)$ is the set of neighbors of node a, and C is a decay factor between 0 and 1.

We compute the SimRank distances and convert similarities to distances by:

$$d(a,b) = 1 - s(a,b)$$

3. At each step, merge the pair of clusters with the smallest distance according to the linkage criterion.

4. Repeat until the desired number of clusters is reached.

After forming clusters, we assign class labels to clusters by mapping each cluster to the most frequent class label among its nodes. The test node's predicted class is then determined by the cluster it belongs to.

Relational Neighbor Classifier The Relational Neighbor Classifier Vojtek & Bieliková (2010); Kajdanowicz (2015); Zhang et al. (2018) is a collective classification algorithm that iteratively updates class membership probabilities based on the labels of neighboring nodes (it resembles Label Propagation a lot but has a few differences). It works by propagates label information through the network.

The algorithm involves the following steps:

1. Initialize conditional class membership probabilities $P^{(0)}$ for all nodes. For labeled nodes v, set:

$$P^{(0)}(v) = e_{y_v}$$

where e_{y_v} is a one-hot vector with 1 at the index corresponding to y_v and 0 elsewhere. For unlabeled nodes, initialize $P^{(0)}(v)$ uniformly:

$$P^{(0)}(v) = \frac{1}{K}\mathbf{1}$$

2. Construct the adjacency matrix A of the graph G.

3. At each iteration t, update the conditional probabilities:

$$P^{(t+1)} = AP^{(t)}$$

4. After each update, re-normalize $P^{(t+1)}$ so that the probabilities sum to 1 for each node.

5. For labeled nodes, reset their probabilities to the initial values to maintain their known labels.

6. Repeat steps 3-5 until convergence, i.e., when the change in probabilities between iterations falls below a threshold.

The final class prediction for each node v is given by:

$$\hat{y}_v = \arg\max_k P_k^{(t)}(v)$$

where $P_k^{(t)}(v)$ is the probability of node v belonging to class k at iteration t.

In our implementation, we focus on classifying one test node at a time. We construct the subgraph containing the test node and its connected training nodes. We initialize the probabilities for the training nodes using their known labels and for the test node with a uniform distribution. We then iteratively update the probabilities until convergence, yielding the predicted class for the test node.

E VALIDATION SCENARIO OVERVIEW

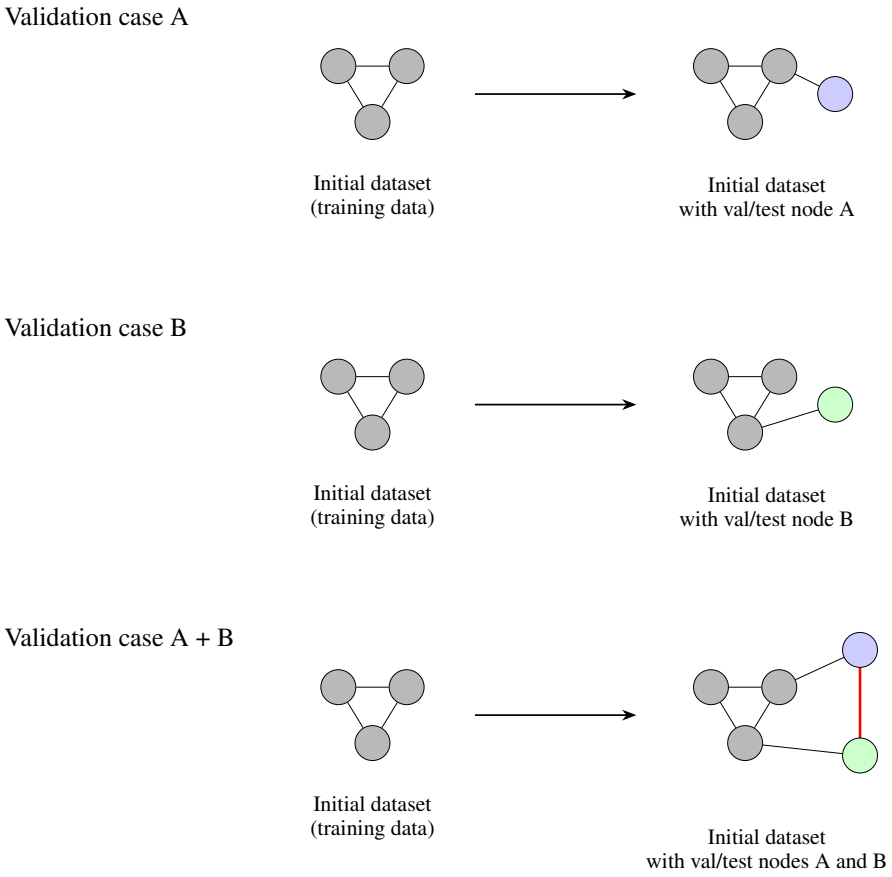

Figure 2: Validation scenario with unlabeled vertices present in the training graph. The newly added node can form edges with both labeled and unlabeled nodes, so the feature generation method must preserve the features of training vertices, remain consistent under changes in graph structure, and correctly account for nodes without class labels. To avoid mutual influence between simultaneously arriving node at inferring stage and potential biases in prediction, we must add exactly one vertex at a time during validation and test phases.

F GNN MODELS

Table 8: In Appendix G we reported performance of four configurations of the models: shallow/narrow (3 layers, hidden size 128), deep/narrow (9, hidden size 128), shallow/wide (3, hidden size 512), and deep/wide (9, hidden size 512). Some entries show larger depth and width than these nominal configurations because of how we measured them. Depth counts every learnable block that produced outputs during a forward pass, including both graph message-passing layers and `Linear` layers. For this reason GIN models appear deeper (for example, the 3-layer GIN becomes 10 and the 9-layer GIN becomes 28), since each `GINConv` contains a two-layer MLP in addition to the classifier. GAT models include a final `Linear` classifier, so the 3- and 9-layer settings register as 4 and 10, respectively. Width is the representative hidden feature size, taken as the modal output channel count across hooked layers, excluding the logits. For multi-head attention with concatenation, the effective width equals `hidden_channels` multiplied by `heads` (for instance, if present 128 with 2 heads gives 256; 512 with 2 heads gives 1024), which explains the rows with 256 and 1024 values. "Trainable" column shows trainable parameter counts. "Activations" column lists the non-linearities between the layers.

Model	**Depth**	**Width**	**Trainable**	**Activations**
GL_GCNConv_3l_128h_w	3	128	17,668	ELU
GL_MLP_3l_128h	3	128	17,668	ELU
GL_SAGEConv_3l_128h	3	128	35,076	ELU
GL_TAGConv_3l_128h_w_k3	3	128	69,892	ELU
GL_GCNConv_9l_128h_w	9	128	116,740	ELU
GL_MLP_9l_128h	9	128	116,740	ELU
GL_SAGEConv_9l_128h	9	128	232,452	ELU
GL_TAGConv_9l_128h_w_k3	9	128	463,876	ELU
GL_GINConv_3l_128h	10	128	83,716	ELU
GL_GINConv_9l_128h	28	128	281,860	ELU
GL_GATConv_3l_128h	4	256	269,060	ELU
GL_GATConv_9l_128h	10	256	1,063,172	ELU
GL_GCNConv_3l_512h_w	3	512	267,268	ELU
GL_MLP_3l_512h	3	512	267,268	ELU
GL_SAGEConv_3l_512h	3	512	533,508	ELU
GL_TAGConv_3l_512h_nw_k3_gnorm	3	512	1,069,060	ELU
GL_TAGConv_3l_512h_nw_k3_gnorm_gelu	3	512	1,069,060	GELU
GL_TAGConv_3l_512h_nw_k3_gnorm_leaky_relu	3	512	1,069,060	LEAKY_RELU
GL_TAGConv_3l_512h_nw_k3_gnorm_relu	3	512	1,069,060	RELU
GL_TAGConv_3l_512h_w_k3	3	512	1,065,988	ELU
GL_TAGConv_3l_512h_w_k3_gnorm	3	512	1,069,060	ELU
GL_TAGConv_3l_512h_w_k3_gnorm_gelu	3	512	1,069,060	GELU
GL_TAGConv_3l_512h_w_k3_gnorm_leaky_relu	3	512	1,069,060	LEAKY_RELU
GL_TAGConv_3l_512h_w_k3_gnorm_relu	3	512	1,069,060	RELU
GL_GCNConv_9l_512h_w	9	512	1,843,204	ELU
GL_MLP_9l_512h	9	512	1,843,204	ELU
GL_TAGConv_3l_512h_w_k3_gnorm_sk	3	512	3,154,948	ELU
GL_GNNM	3	512	3,160,688	ELU
GL_SAGEConv_9l_512h	9	512	3,682,308	ELU
GL_TAGConv_9l_512h_w_k3	9	512	7,360,516	ELU
GL_GINConv_3l_512h	10	512	1,317,892	ELU
GL_GINConv_9l_512h	28	512	4,469,764	ELU
GL_GATConv_3l_512h	4	1024	4,221,956	ELU
GL_GATConv_9l_512h	10	1024	16,835,588	ELU

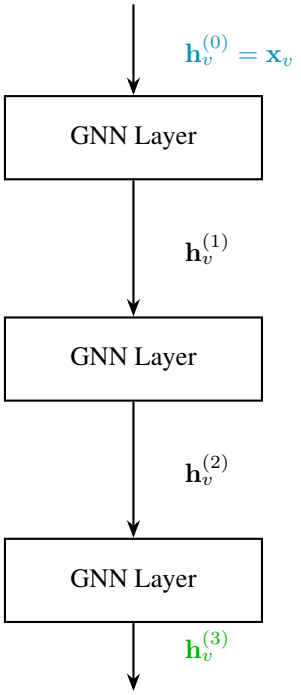

Figure 3: In our setting we simply stack several (possibly different) GNN layers sequentially: the node's representation $\mathbf{h}_v^{(k)}$ after layer k becomes the input for the next layer.

G DEPTH AND WIDTH OF MODELS

Both GNNs and MLPs were tested with four architecture variants:

- **shallow, narrow**: 3 layers, hidden dimension 128.
- **deep, narrow**: 9 layers, hidden dimension 128.
- **shallow, wide**: 3 layers, hidden dimension 512.
- **deep, wide**: 9 layers, hidden dimension 512.

The best-performing setting, for both GNNs and MLPs, is the shallow, wide variant (three layers with hidden size 512), which performs best on all datasets and is shown in Figure 4. This result accords with the well-known oversmoothing problem Oono & Suzuki (2020); Rusch et al. (2023a); Wu et al. (2022; 2023) in deep GNNs: with too many layers (without skip connections) embeddings tend to wash out across the graph.

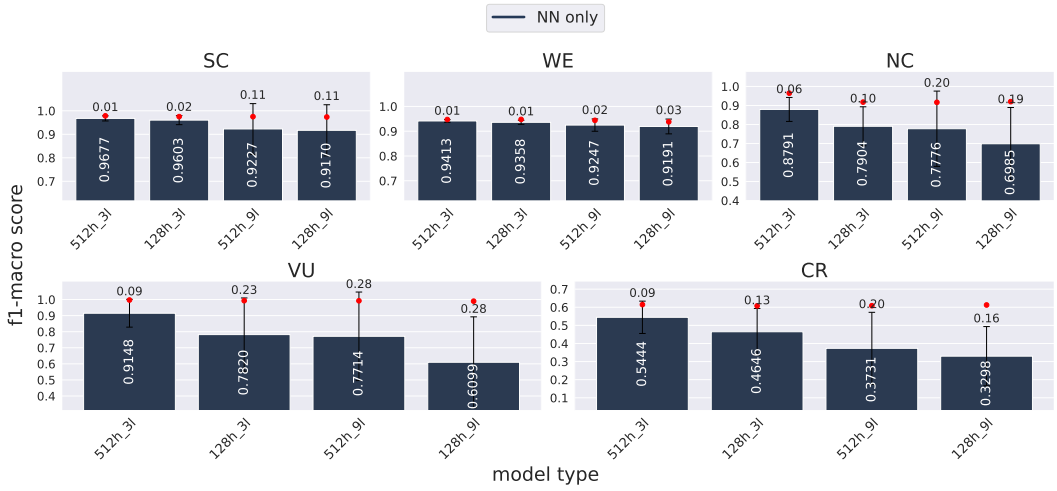

Figure 4: Effect of neural network width and depth on classification performance. Bars show mean f1-macro scores for all selected GNN and MLP models trained on the five fully labeled datasets, grouped into four architecture categories by layer depth and hidden size. Black error bars indicate the standard deviation across models within each category, and red dots mark the best-performing model in that category. Across all datasets, shallow and wide networks (3 layers, hidden size 512) consistently achieve the highest accuracy, demonstrating that wider but less deep architectures are most suitable for this classification task.

H OTHER DATASETS WITHOUT ARTIFICIAL MASKED NODES

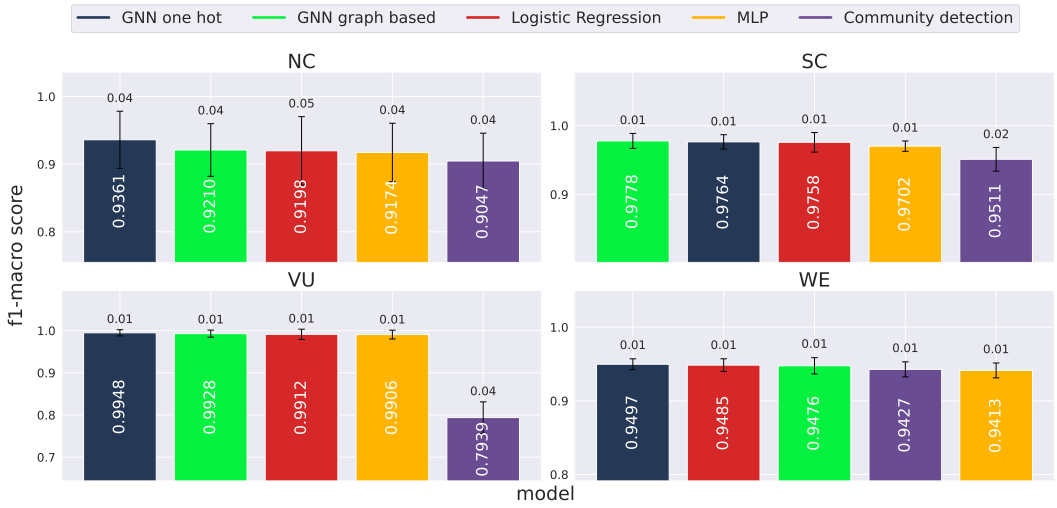

Figure 5: Performance comparison on supplementary datasets (NC, SC, VU and WE). The scores only for the best models in each category are shown. Results demonstrate that our best-performing Graph Neural Network trained with one-hot features mostly outperforms the top graph-based feature models.

I SIMULATIONS

I.1 ARTIFICIAL MASKING

Four of the five real graphs except ES are fully labeled, and to evaluate how our model behaves when labels are partially missing on population structures besides ES we perform a masking procedure that simulates the presence of unlabeled vertices. We split vertices into train, validation, and test sets exactly as described in the **Methods** section in the paragraph "Models and training regimes" for all five datasets, including the labeled-only portion of ES, with the following modification. Within the 60 percent training portion we select a fixed core subset equal to 25 percent of the training vertices. This core subset preserves the original class balance and remains labeled for all masking percentages and across all ten splits. The remaining 75 percent of the training vertices are considered removable. For a masking percentage $p \in \{0, 25, 50, 75, 100\}$ we randomly select p percent of these removable vertices and restore their labels for that experiment, while all remaining removable vertices stay unlabeled during training, validation, and test. The core labeled subset remains unchanged for all masking percentages. When p is 0 percent, only the fixed labeled core participates in training for every split, so all ten splits are identical. When p is 100 percent, all removable vertices are unlabeled in every split, and because every split uses the entire set of removable unlabeled vertices without any sampling variability, all ten splits are again identical. These two boundary cases therefore produce zero variation across splits, which explains zero rounded standard deviations of in Table 9.

In this section, we are focusing on ES, as this dataset is the primary object of our study. Corresponding analyses for the other four fully labeled graphs (SC, WE, NC, and VU) are depicted in Figure 6. The rows in Table 9 list the best-performing architectures within each model family. For the GNN one hot row the optimal architecture for all label-removal percentages is `GL_TAGConv_3l_512h_w_k3_gnorm`. For the GNN graph based row the strongest architecture depends on the percentage of unlabeled training vertices, namely `GL_SAGEConv_3l_512h` at 0 percent, `GL_GINConv_3l_512h` at 25 and 50 percent, and `GL_TAGConv_3l_512h_w_k3_gnorm` at 75 and 100 percent. For MLP models the best configuration at 0 percent unlabeled vertices is `GL_MLP_3l_128h`. For all higher percentages the MLP entries are left blank because this model family cannot exploit unlabeled vertices. Community detection algorithms are omitted in this setting, consistent with their treatment in the Appendix J.

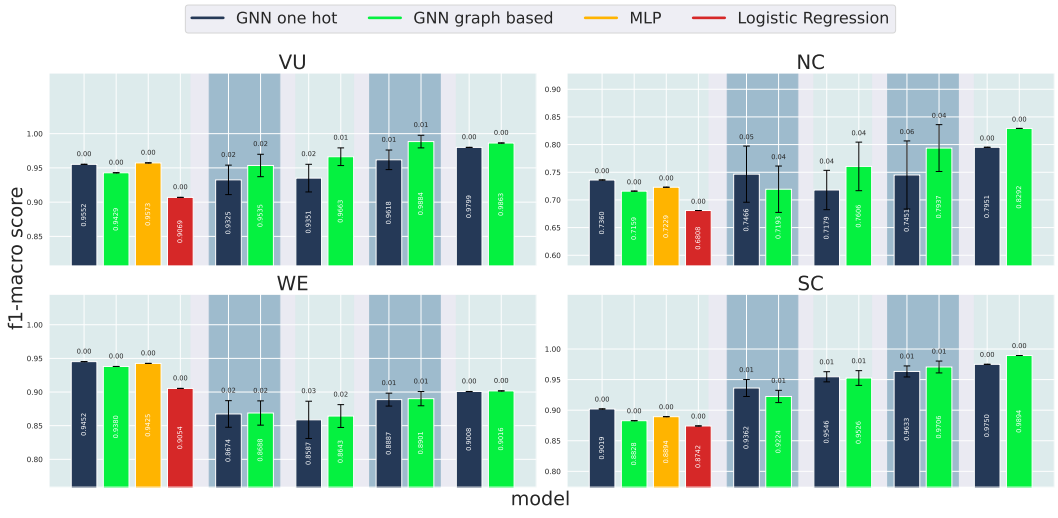

Figure 6: Performance comparison on supplementary datasets (NC, SC, VU and WE). The scores only for the best models are shown. Results demonstrate that our best-performing Graph Neural Network trained with one-hot features significantly outperforms the top graph-based feature models.

The results in Table 9 show that GNNs improve consistently as the proportion of unlabeled training vertices increases. In the WE and NC datasets we additionally identify vertices whose neighborhoods consist entirely of either unlabeled or labeled neighbors. On WE, for vertices with only

Table 9: ES with artificial label masking. F1-macro across masking rates. All values are averaged over ten splits.

Model	0%		25%		50%		75%		100%	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
GNN one hot	0.4651	0.00	**0.5012**	0.02	**0.5356**	0.01	**0.5610**	0.01	**0.5840**	0.00
GNN graph based	0.4345	0.00	0.4715	0.01	0.5074	0.01	0.5306	0.01	0.5528	0.00
MLP	0.4659	0.00	—	—	—	—	—	—	—	—
LR	0.4279	0.00	—	—	—	—	—	—	—	—

unlabeled neighbors, the best GNN achieves an f1-macro of 0.8348 over 191 such vertices, and for vertices connected exclusively to labeled neighbors it reaches 1.0 over 26 vertices. On NC, the corresponding values are 0.5306 over 25 vertices with only unlabeled neighbors and 0.5 over 4 with only labeled neighbors. All remaining vertices in these datasets have mixed neighborhoods.

These results illustrate how message passing leverages unlabeled intermediates. Even when a vertex has no direct labeled neighbors, information can propagate through unlabeled vertices.

I.2 Global sparsity performance

We study robustness to global sparsity on both real and simulated data (see Appendix L) built on the Volga–Ural (VU) population graph. VU is our densest dataset and therefore allows us to probe a broad range of sparsity parameters before the graph becomes disconnected. For the real-data experiment we vary the fraction ρ of vertices used for training, while keeping the validation and test sets unchanged. The training fraction ranges from $\rho = 0.01$ up to $\rho = 1.0$. At each value of ρ we evaluate the best model in each family: a GNN with one hot features (`GL_TAGConv_3l_512h_w_k3_gnorm`), the same architecture with graph based features, logistic regression (`GL_LR`), and an MLP (`GL_MLP_3l_512h`). For every ρ we train on ten random train/validation/test splits and report mean f1-macro with one-standard-deviation bands. Figure 7 shows that all learned models improve rapidly as ρ increases from 0.01 to about 0.1, after which their curves flatten and approach f1-macro values near one. The GNN with one hot features is consistently strongest, followed closely by the graph based GNN and logistic regression, while the MLP lags slightly behind.

To complement this experiment on real data we perform a second global sparsity experiment on simulated VU graphs in which we scale all edge probabilities while keeping class proportions close to those in the original VU dataset. Let $P_{c,c'}$ denote the estimated probability that a node of class c connects to class c'. For a scalar $\alpha \in (0, 1]$ we define the down-scaled matrix

$$P^{(\text{new})}_{c,c'} = \alpha P_{c,c'}, \tag{9}$$

and generate a new graph using $P^{(\text{new})}$ together with the original mean IBD segment lengths. For each value of α we simulate one graph, train the same set of models as in the real-data experiment, and average f1-macro over ten random splits. As shown in Figure 8, the learned models again benefit monotonically from increasing α: as the graph becomes denser their accuracy improves and stabilizes at high f1-macro, with the one hot GNN slightly ahead of the graph based GNN and logistic regression and the MLP just below them.

Taken together, these two global sparsity experiments indicate that the one hot GNN is most robust to reductions in training data and edge density on VU, with the graph based GNN and logistic regression close behind, whereas the MLP degrades more strongly when the graph is sparse or training data are scarce.

I.3 Class imbalance performance

We assess the impact of class imbalance on model performance using simulated graphs based on the Eastern Slavs (ES) population structure. These graphs preserve the four population labels present in the real ES dataset—Southern Russians, Ukrainians, Northern Russians, and Belarusians—but we systematically modify their class sizes. For each simulated graph we evaluate the best model

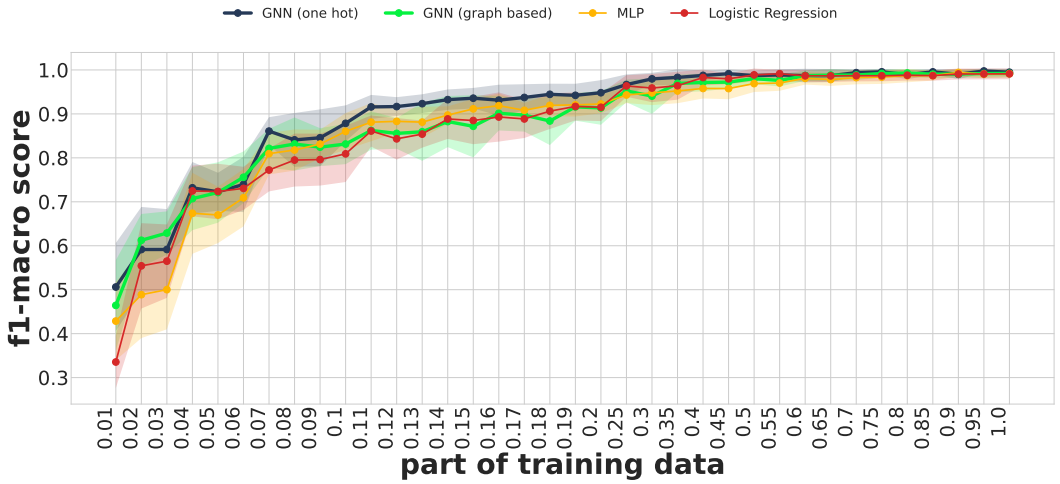

Figure 7: Performance on the VU graph as a function of the fraction ρ of vertices used for training. Curves show mean f1-macro over ten random data splits for the best model in each family: GNN with one hot features, GNN with graph based features, MLP, and logistic regression. Shaded regions denote one standard deviation.

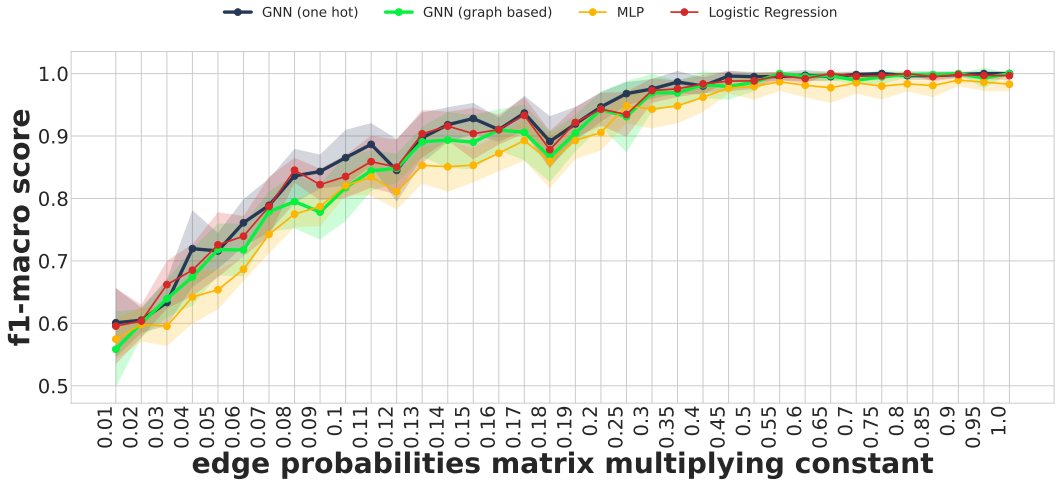

Figure 8: Uniform scaling of all class-pair edge probabilities in simulated VU graphs. The horizontal axis shows the multiplier α applied to the connection probability matrix P as in equation 9, and the vertical axis shows mean f1-macro over ten random splits for the best model in each family. The GNN with one hot features is consistently strongest, followed by the graph based GNN and logistic regression. The MLP is slightly weaker.

from each family: a GNN with one hot features (`GL_TAGConv_3l_512h_w_k3_gnorm`), a GNN with graph based features using the same architecture, logistic regression (`GL_LR`), and an MLP (`GL_MLP_3l_512h`).

To isolate the effect of class imbalance from the effect of sample size, we construct a family of six simulated ES graphs in which the total number of vertices remains nearly constant while we gradually rebalance the four population groups. Let $n_k^{(0)}$ denote the original number of vertices in population k, with total $N = \sum_k n_k^{(0)}$, and let $n_{\min}$ and $n_{\max}$ be the minimum and maximum of

these counts. For each interpolation step $t \in \{0, \ldots, 5\}$ we first choose a lower bound

$$a_t = n_{\min} + \frac{t}{5}\big(n_{\max} - n_{\min}\big),$$

so that $a_0 = n_{\min}$ and $a_5 = n_{\max}$. We then form a sorted vector of intermediate target sizes

$$u_k^{(t)} = a_t + \frac{k}{3}\big(n_{\max} - a_t\big), \quad k = 0, 1, 2, 3,$$

which is linearly spaced between a_t and $n_{\max}$. The smallest element of this vector is assigned to the population that was smallest in the original ES distribution, the next smallest element to the second smallest population, and so on, so that classes keep their rank ordering while moving toward balance. Finally, we renormalize these target sizes so that the total number of vertices matches the original total N by setting

$$n_k^{(t)} = \left\lfloor \frac{u_k^{(t)}}{\sum_k u_k^{(t)}} N \right\rfloor,$$

where $\lfloor \cdot \rfloor$ denotes the floor operator. For each step we then generate a new random graph using the same class-pair edge probability matrix and mean IBD segment lengths estimated from the real ES dataset, but with the new class sizes $n_k^{(t)}$. The resulting class sizes are summarized in Table 10. From left to right in Fig. 9 the six blocks of bars correspond to these six interpolation steps, moving from the original imbalanced configuration (Step 0) to an exactly balanced configuration (Step 5).

Table 10: Simulated ES graphs used for the class-balance interpolation. Each column corresponds to one configuration in Fig. 9, from the original imbalanced setting (Step 0) to the fully balanced setting (Step 5). Entries are numbers of vertices per population group.

Population group	**Step 0**	**Step 1**	**Step 2**	**Step 3**	**Step 4**	**Step 5**
Southern Russians	1626	1505	1400	1309	1229	1158
Ukrainians	1002	1042	1077	1107	1134	1158
Northern Russians	1313	1273	1238	1208	1181	1158
Belarusians	690	811	916	1007	1087	1158
Total vertices	4631	4631	4631	4631	4631	4632

Across this interpolation the model families using learned parameters—GNNs with one hot and graph based features, logistic regression, and MLP—show only modest variation in f1-macro. Their scores remain close to one another, typically around 0.70–0.72 with small standard deviations over ten splits, and no strong systematic trend appears as the class distribution is rebalanced. Overall, these simulations indicate that the models we consider are robust to substantial changes in population proportions.

J LIMITATIONS OF THE MODELS

During testing, Logistic regression (LR) and multi layer perceptron (MLP) models, act as classifiers built on top of the heuristics because they operate solely on precomputed graph based node features. These features already encapsulate all statistical quantities used by heuristic methods, such as the number of labeled neighbors, edge weights, and segment counts, combined into a single feature vector. While GNNs also utilize such feature representations, they additionally incorporate message passing over the graph structure, allowing them to capture topological relationships and contextual dependencies that heuristics, LR, and MLPs cannot.

Some community detection algorithms, such as label propagation or relational label classifier, can also use both node features and graph topology. However, these approaches generally exhibit lower predictive power compared to GNNs, as demonstrated in the **Results** section of this work. Moreover, traditional clustering methods, including spectral and agglomerative algorithms, can assign labels to a single newly added vertex by analyzing its connections to the labeled subgraph, but when multiple unlabeled vertices are introduced, they tend to form separate clusters that may not correspond to known populations. In such cases, these methods often fail to assign valid class labels or may even

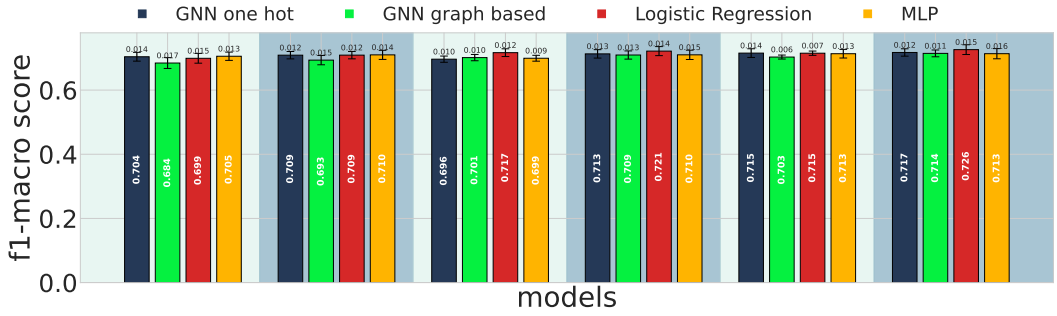

Figure 9: Class-balance interpolation on simulated ES graphs. Each block of bars corresponds to one configuration from Table 10, moving from a strongly imbalanced setting (Step 0, left) toward a fully balanced setting (Step 5, right). Bars show mean f1-macro scores with error bars denoting one standard deviation over ten splits for the best model in each family: GNN with one hot features and GNN with graph based features with `GL_TAGConv_3l_512h_w_k3_gnorm` architecture, logistic regression (`GL_LR`), and MLP (`GL_MLP_3l_512h`).

omit entire groups. Therefore, clustering-based techniques are applied only in the one-vertex-at-a-time scenario, where a single unlabeled node is classified based on its links to the labeled core of the graph.

In contrast, GNNs with all types of node features overcome these limitations by combining feature information with the full graph topology through message passing. This enables them to make robust and consistent predictions even in the presence of numerous unlabeled vertices and complex graph structures.

K Runtime and Memory

To fully replicate all computations performed in this study, approximately three weeks of computation time are required on a server equipped with two A100 GPUs (80 GB each) and a CPU with 16 cores and 128 GB RAM. The provided supplementary repository includes code designed for efficient training of these models, which can be executed on any GPU-enabled cluster environment with parallel training enabled.

This research was supported in part through computational resources of HPC facilities at HSE University. All experiments were performed on the cHARISMa high performance computer (Kostenetskiy et al., 2021).

L Simulator

Let K be the total number of classes. We label the classes as:

$$\{P_0, P_1, \ldots, P_{K-1}\}.$$

We let

$$p = \{p_{ij}\}$$

be the matrix of probabilities of existence of links between classes i and j computed from real graphs. Furthermore, we let

$$\theta = \{\theta_{ij}\}$$

be the matrix of mean weights for edges between classes i and j computed from real graphs. Finally, we denote by α the offset (here $\alpha = 6.0$) that shifts the exponential distribution because minimal edge weight must be 6.0 in accordance with real data.

Symmetrization function. A small utility function,

$$\texttt{symmetrize}(M) \;=\; M + M^T \;-\; \mathrm{diag}(M),$$

takes a square matrix M and ensures that the resulting matrix is symmetric by reflecting the lower and upper triangular parts around the diagonal.

Matrix generation. Let n_i be the size of class i, so we have a list of population sizes $\{n_0, n_1, \ldots, n_{K-1}\}$. Initially, we generate two matrices: `counts` (edges between nodes) and `sums` (edge weights for each edge). We organize generation into generation of blocks. Because we have K classes, each pair (i, j) corresponds to a block of size $n_i \times n_j$. As a result, two large matrices are formed:

$$\texttt{counts} \;=\; \begin{bmatrix} B_{00} & B_{01} & \cdots & B_{0,K-1} \\ B_{10} & B_{11} & \cdots & B_{1,K-1} \\ \vdots & \vdots & \ddots & \vdots \\ B_{K-1,0} & B_{K-1,1} & \cdots & B_{K-1,K-1} \end{bmatrix},$$

$$\texttt{sums} \;=\; \begin{bmatrix} S_{00} & S_{01} & \cdots & S_{0,K-1} \\ S_{10} & S_{11} & \cdots & S_{1,K-1} \\ \vdots & \vdots & \ddots & \vdots \\ S_{K-1,0} & S_{K-1,1} & \cdots & S_{K-1,K-1} \end{bmatrix},$$

where each B_{ij} and S_{ij} has shape $n_i \times n_j$.

Each element (submatrix) B_{ij} of matrix `counts` corresponds to a matrix that represents link existence (0 or 1) for all individuals in population i to all individuals in population j. An analogous structure with elements S_{ij} of array `sums` stores edge weights for those pairs.

For each pair of classes (i, j) with $0 \le j \le i < K$ (other submatrices are zeros), we do the following:

1. Get a vector with a size of $n_i \times n_j$ (vector of values 0 and 1 that represent existence or lack of edge between vertices of two population groups)
$$b = (b_1, \ldots, b_{n_i \times n_j}), \quad b_\ell \sim \mathrm{Bernoulli}\big(p_{ij}\big).$$
2. Let
$$T \;=\; \sum b$$
to get the amount of edges.
3. Get a vector with a size of T (sample edge weights only for generated edges)
$$y = \big(y_1, \ldots, y_T\big), \quad y_k \sim \mathrm{Exponential}\big(\lambda = 1/\theta_{ij}\big), \quad k = 1, \ldots, T.$$
and set
$$s = y + \alpha.$$
4. Reshape b into the matrix $B_{ij} \in \{0,1\}^{n_i \times n_j}$. Form the matrix $S_{ij} \in \mathbb{R}_{\ge 0}^{n_i \times n_j}$ by placing the weights s_T in the positions corresponding to the entries $b_\ell = 1$ (in index order) and zeros elsewhere.
5. If $i = j$, zero the diagonal and above to avoid self-loops:
$$B_{ii} \leftarrow \mathrm{tril}\big(B_{ii}, -1\big), \quad S_{ii} \leftarrow \mathrm{tril}\big(S_{ii}, -1\big).$$

Finally, we symmetrize the entire matrix `counts` and `sums` using `symmetrize` function, so the final `counts` and `sums` are symmetric.

Storing the simulation output. We write out the graph into a CSV file. Specifically, for each pair of individuals (i, j) that has edge, we store:

$$(\texttt{node\_id1}, \texttt{node\_id2}, \texttt{label\_id1}, \texttt{label\_id2}, \texttt{ibd\_sum}, \texttt{ibd\_n}),$$

where `ibd_sum` corresponds to the edge weight, and `ibd_n` is number of IBD segments which is always 1 because we can not model it yet during simulation.

M GRAPH STATISTICS

In this section, we show graph parameters for the five primary datasets (single graphs) used in this study with the populations grouped by regions of the world: Northern Caucasus (NC), Western Europe (WE), Volga-Ural region (VU), Scandinavia (SC) and Eastern Slavs (ES) in both labeled only and complete versions. For each dataset, we computed (see Table 11) and visualized a variety of graph metrics (e.g., degrees, clustering coefficients, shortest paths; see supplemenary pictures 10, 13, 11, 12, 14, 15, 17, 18, 19, 20, 21, 22, 23) to gain preliminary insight into the population genetics behind each region. All our graphs are undirected, have no multiple edges, and isolated vertices.

Parameter	NC	SC	WE	VU	ES (labeled only)	ES (full)
Total number of unique nodes	614	1086	3857	1642	4633	230887
Total number of unique edges	6297	17733	21941	169379	80635	6802907
Number of multi-edges	0	0	0	0	0	0
Number of self-loops	0	0	0	0	0	0
Number of isolated nodes	0	0	0	0	0	0
Is planar	No	No	No	No	No	No
Is Eulerian	No	No	No	No	No	No
Is semi-Eulerian	No	No	No	No	No	No
Is tree	No	No	No	No	No	No
Is forest	No	No	No	No	No	No
Is regular	No	No	No	No	No	No
Density	0.0334	0.03	0.003	0.1257	0.0075	0.0003
Is connected	No	Yes	No	Yes	Yes	Yes
Is directed	No	No	No	No	No	—
Is homogeneous	Yes	Yes	Yes	Yes	Yes	—

Number of connected components	2	1	2	1	1	1
Number of nodes in first component (largest)	611	—	3781	—	—	—
Number of nodes in second component (smallest)	3	—	3	—	—	—
Radius	7	4	9	2	3	—
Diameter	14	6	17	3	5	—
Maximum degree of nodes	80	92	55	415	107	4087
Mean degree of nodes	20.60	32.66	11.58	206.3081	34.8089	58.93
Minimum degree of nodes	1	1	1	4	1	1
Transitivity coefficient	0.3443	0.1417	0.0332	0.2227	0.0197	0.0012
Global efficiency coefficient	0.3482	0.4169	0.251	0.5617	0.3764	—
Local efficiency	0.3995	0.2837	0.0371	0.5853	0.0307	—
Degree assortativity coefficient	0.4646	0.3767	0.3528	0.3406	0.1444	−0.8307
Class assortativity coefficient	0.6209	0.6968	0.8939	0.3805	0.1871	—

Average clustering	0.2544	0.1019	0.0286	0.1922	0.0192	0.0232
Class partition modularity (no weights)	0.4973	0.3321	0.2609	0.1917	0.1281	—
Average shortest path length	3.4609	2.602	4.5377	1.8809	2.7629	—
Weighted average shortest path length	41.0455	25.3954	44.0865	16.8953	24.5403	—
Maximum degree centrality	0.1311	0.0847	0.0145	0.2528	0.0231	0.0177
Maximum eigenvector centrality	0.1209	0.1115	0.0976	0.0528	0.0505	0.0323
Maximum closeness centrality	0.4045	0.4601	0.294	0.5717	0.3624	—
Maximum betweenness centrality	0.044	0.0108	0.0169	0.0013	0.0027	—

Table 11: Basic statistics of the graphs. In case there are multiple components, parameters are computed for the largest component only. For ES (full) graph properties were calculated with GPU but some properties couldn't be calculated withing a reasonable time due to the size of the graph.

The data presented in Table 12 highlight considerable differences in the number of disconnected graph components on train sets across various geographical regions.

Table 12: Number of connected components of reference graphs across datasets.

Dataset	Components	Std. Dev.
SC	3.3	1.42
WE	177.5	11.32
NC	18.2	3.34
ES	2.4	0.66
VU	1.0	0.00

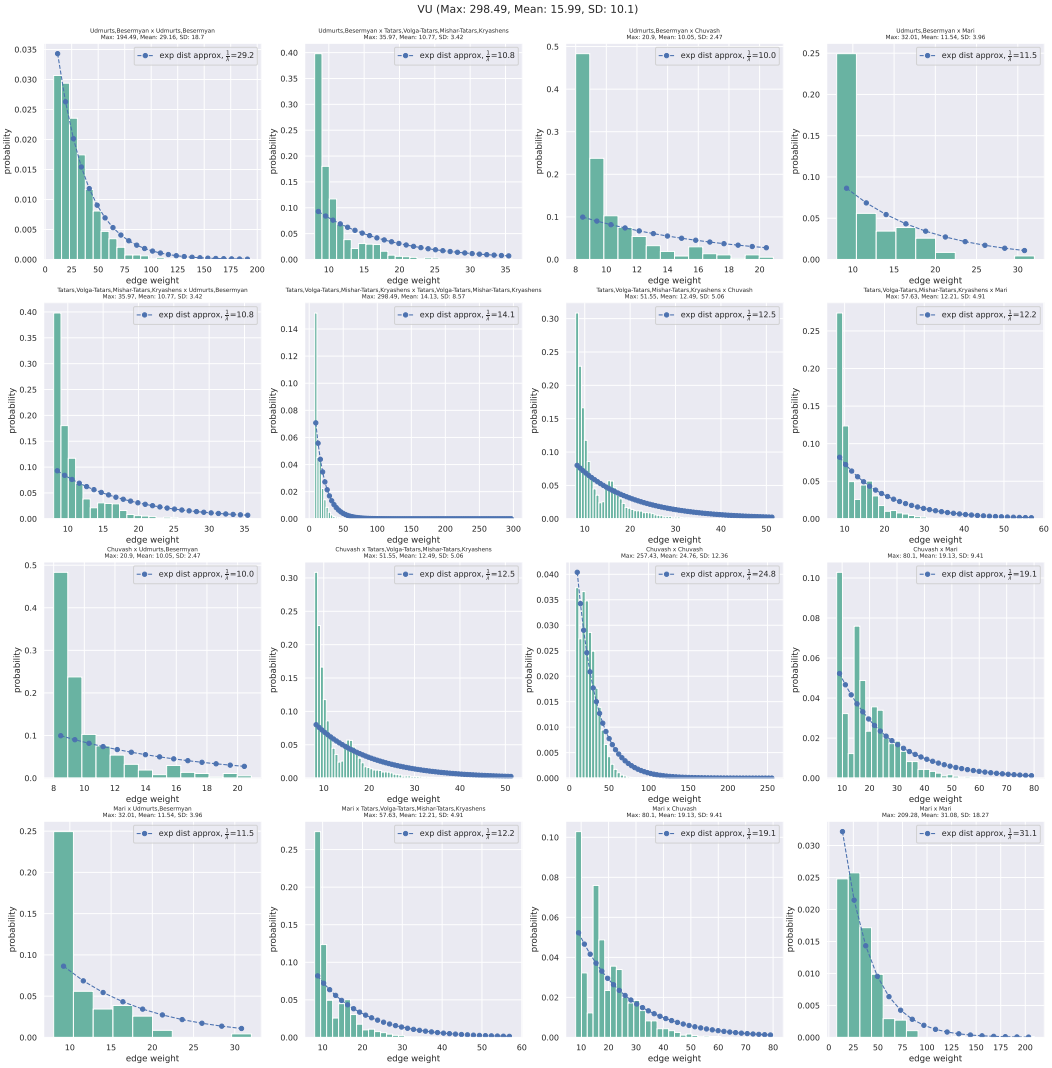

Figure 10: Edge weight distribution for VU dataset.

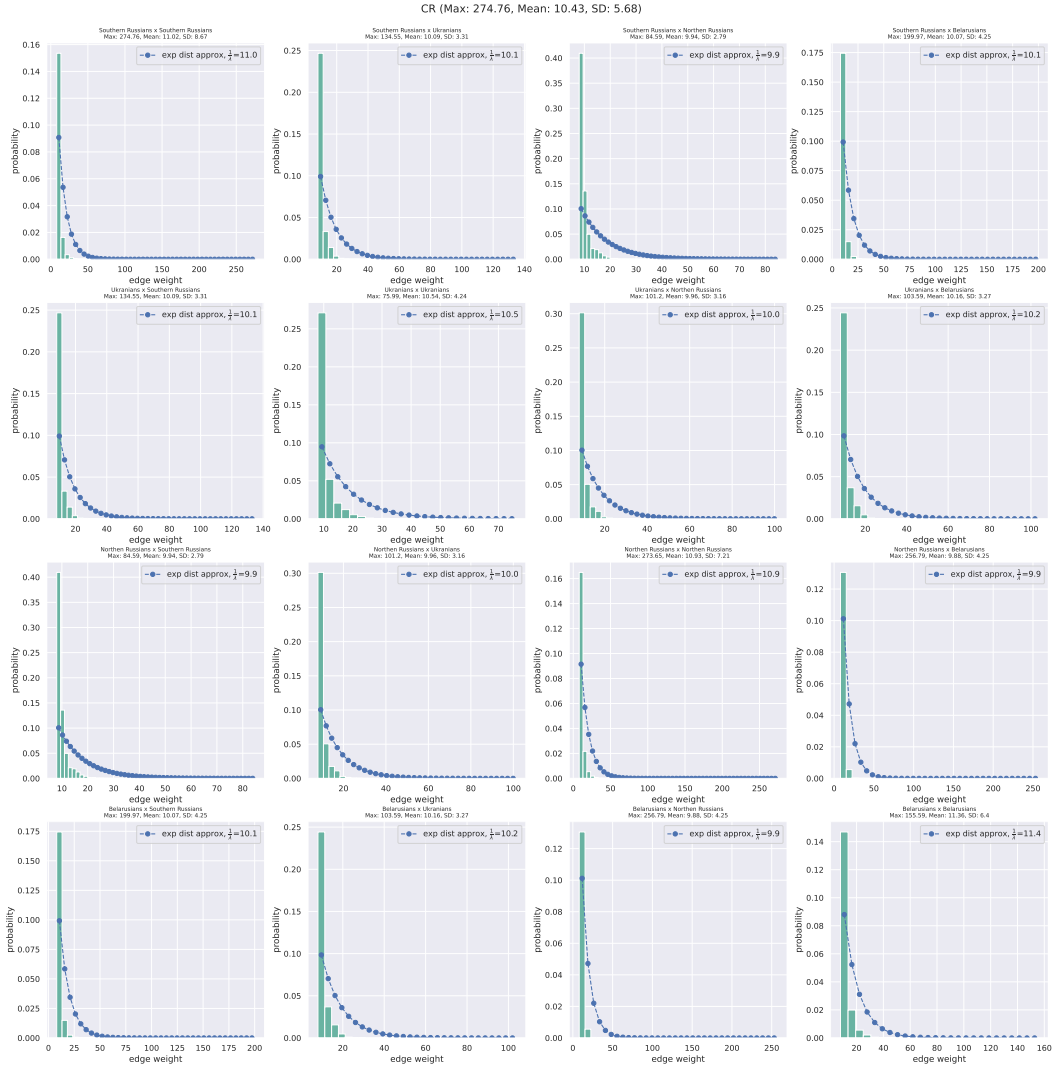

Figure 11: Edge weight distribution for ES dataset (excluding unlabeled vertices).

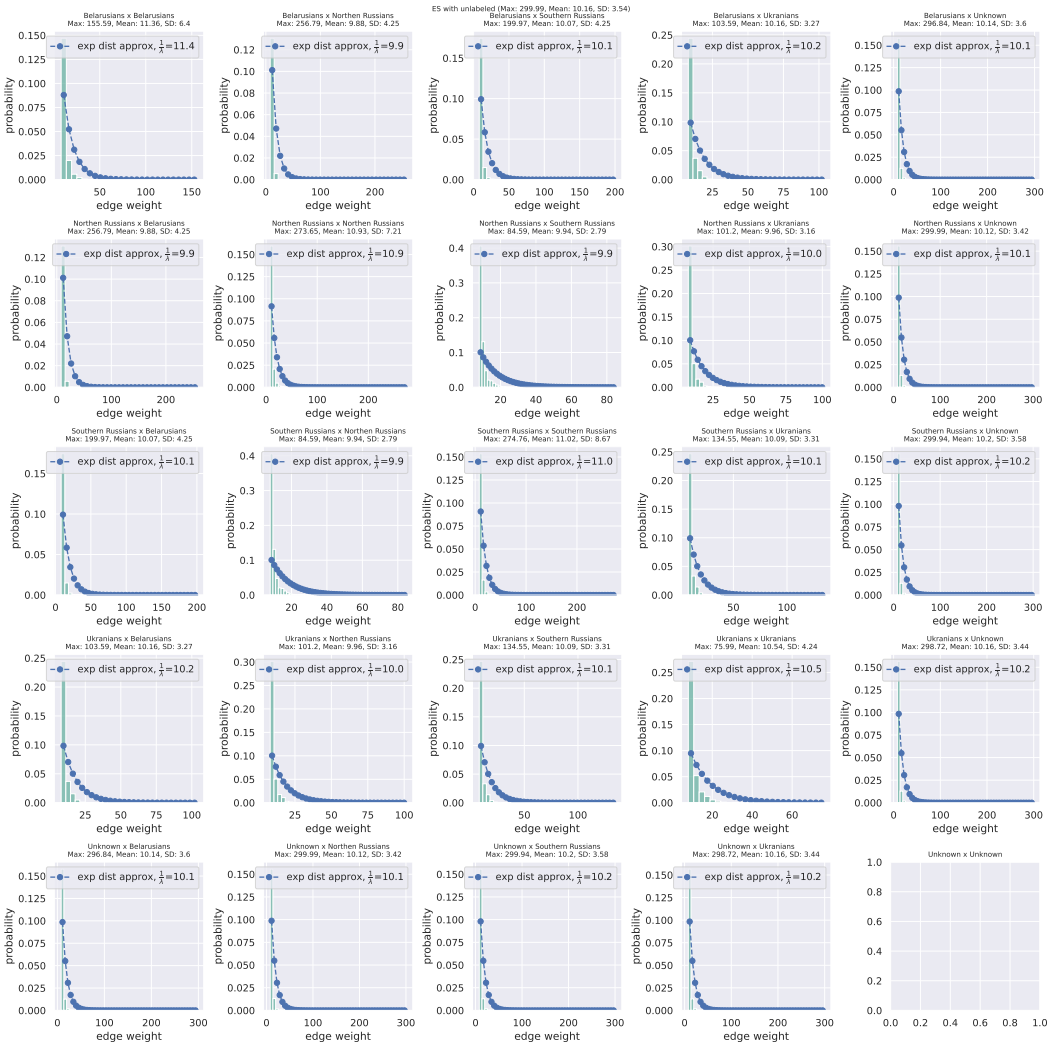

Figure 12: Edge weight distribution for ES dataset (with unlabeled vertices).

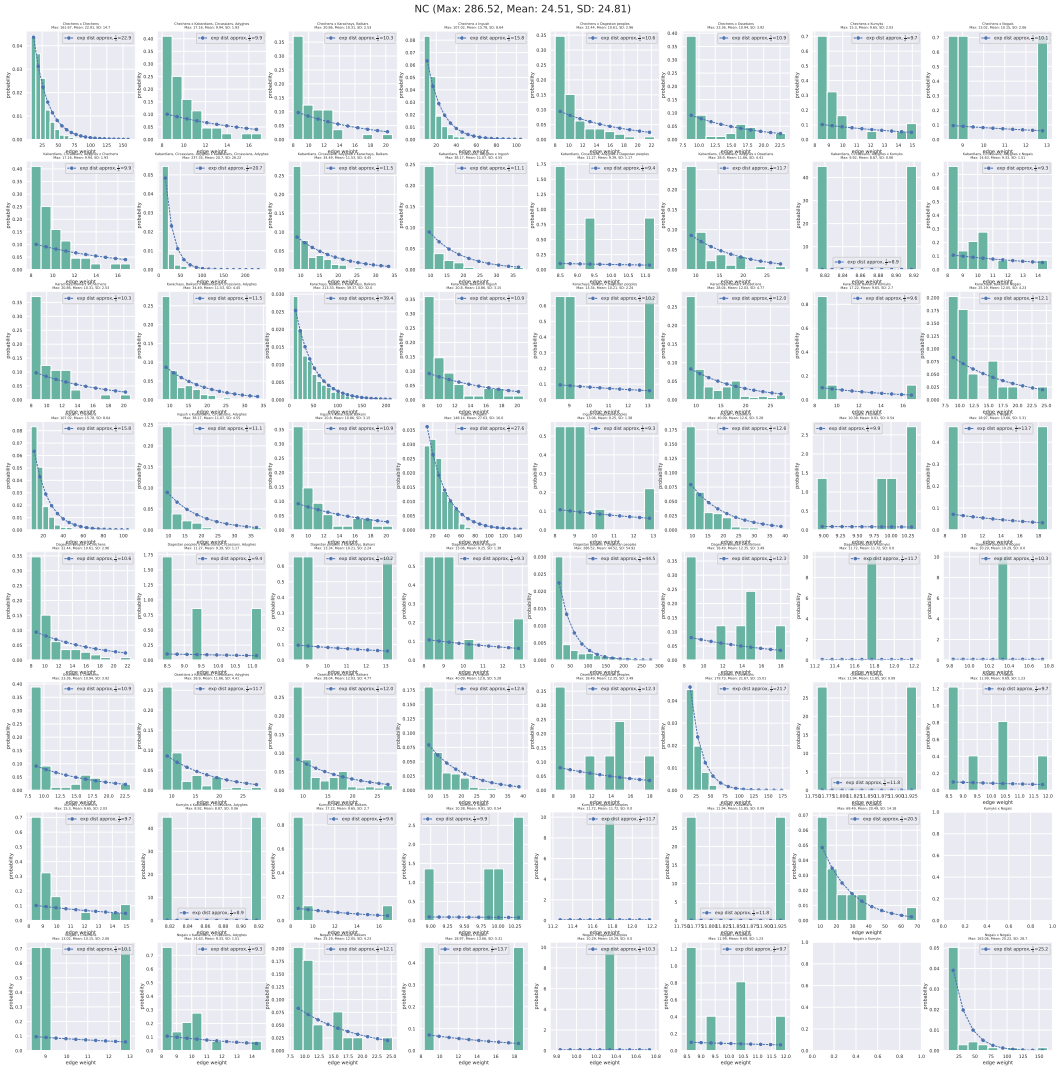

Figure 13: Edge weight distribution for NC dataset.

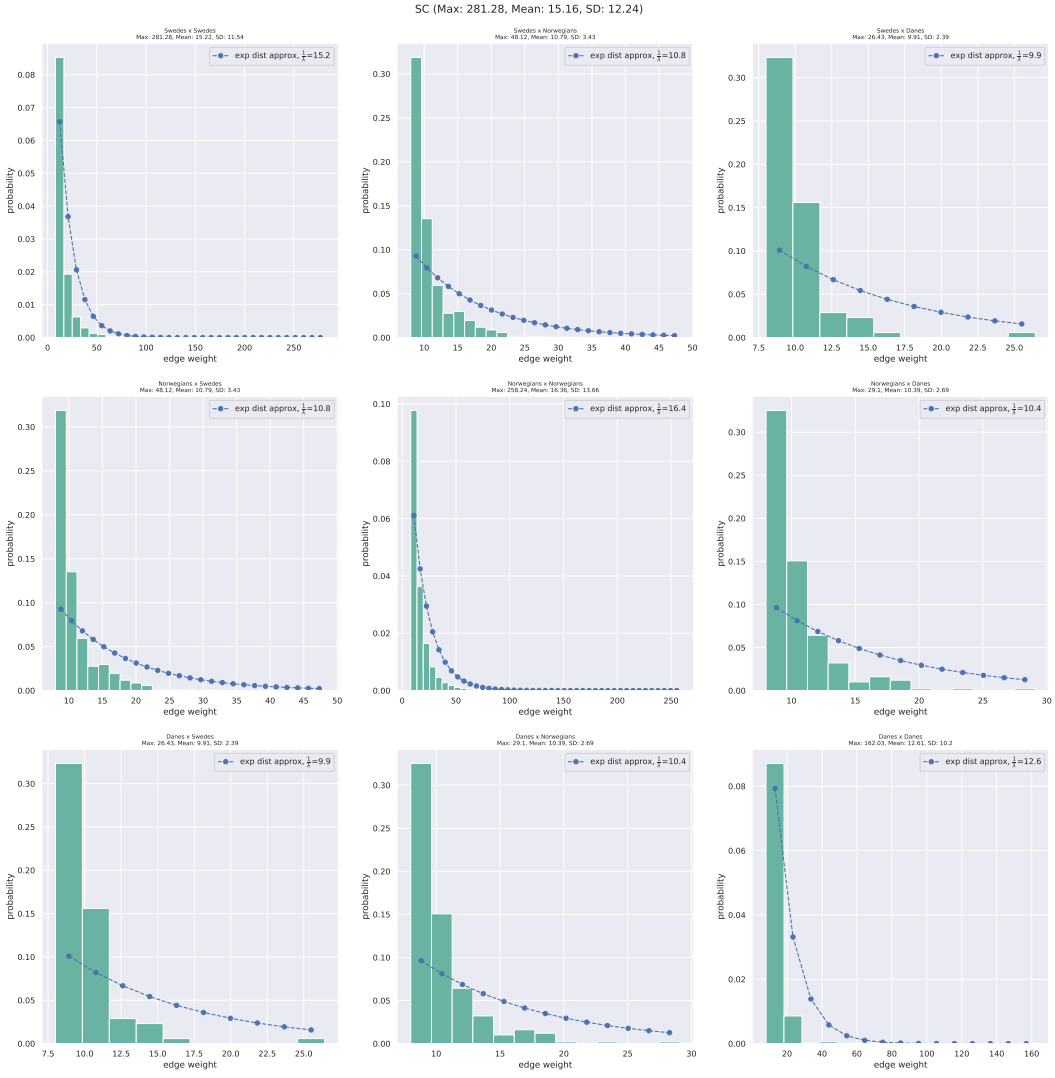

Figure 14: Edge weight distribution for SC dataset.

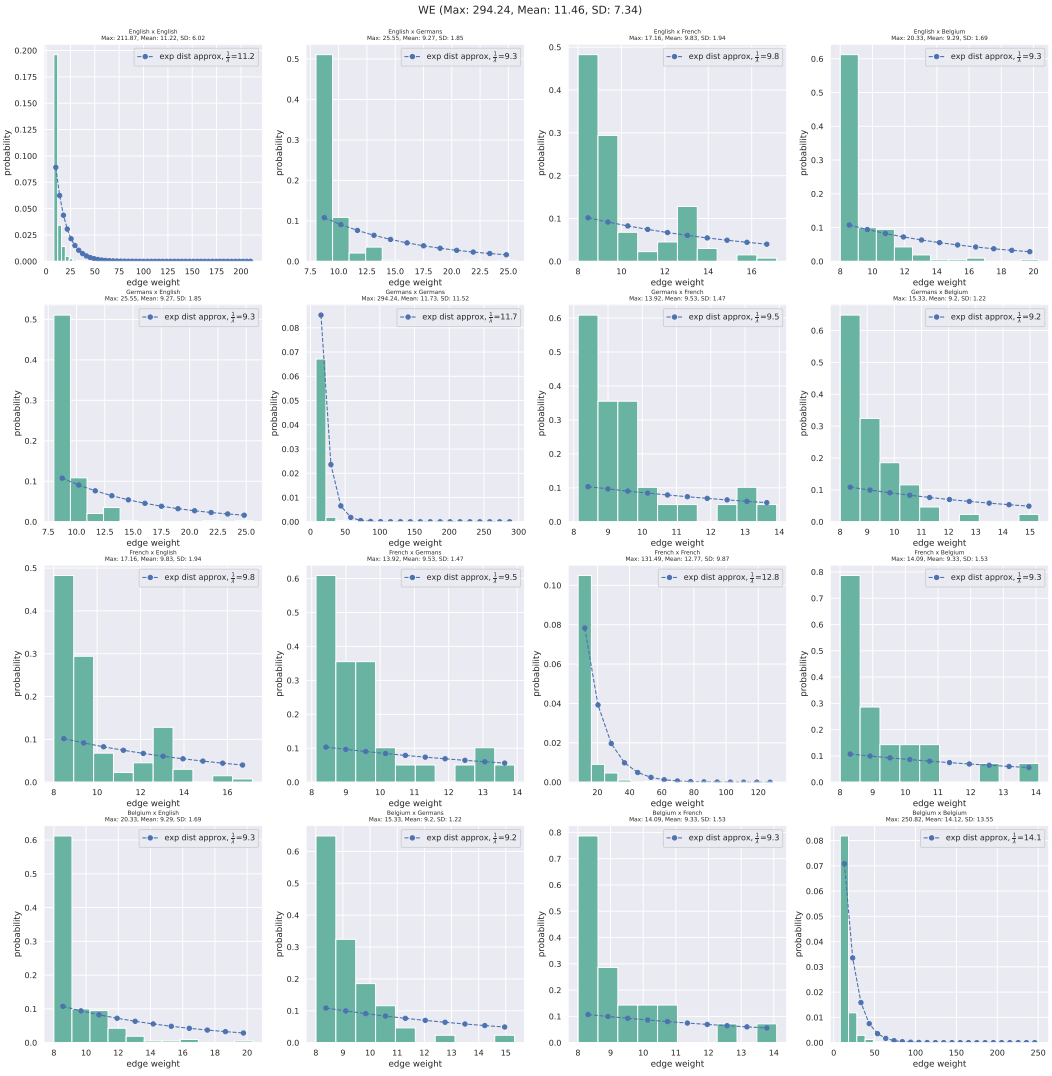

Figure 15: Edge weight distribution for WE dataset.

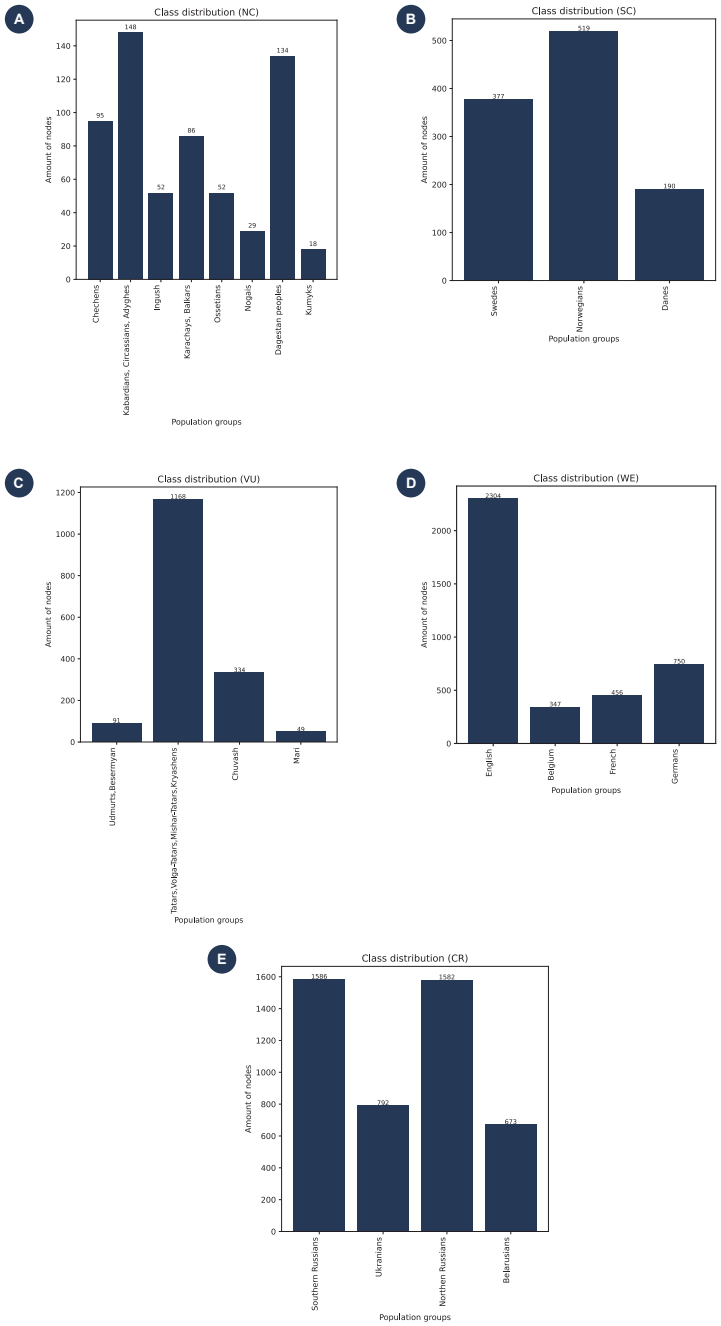

Figure 16: Distribution of vertex counts across population classes in each dataset: (A) Northern Caucasus populations, (B) Scandinavian populations, (C) Volga-Ural region populations, (D) Western European populations, (E) Eastern Slavs populations (excluding unlabeled vertices). Each bar represents the number of vertices belonging to a particular class within the corresponding graph.

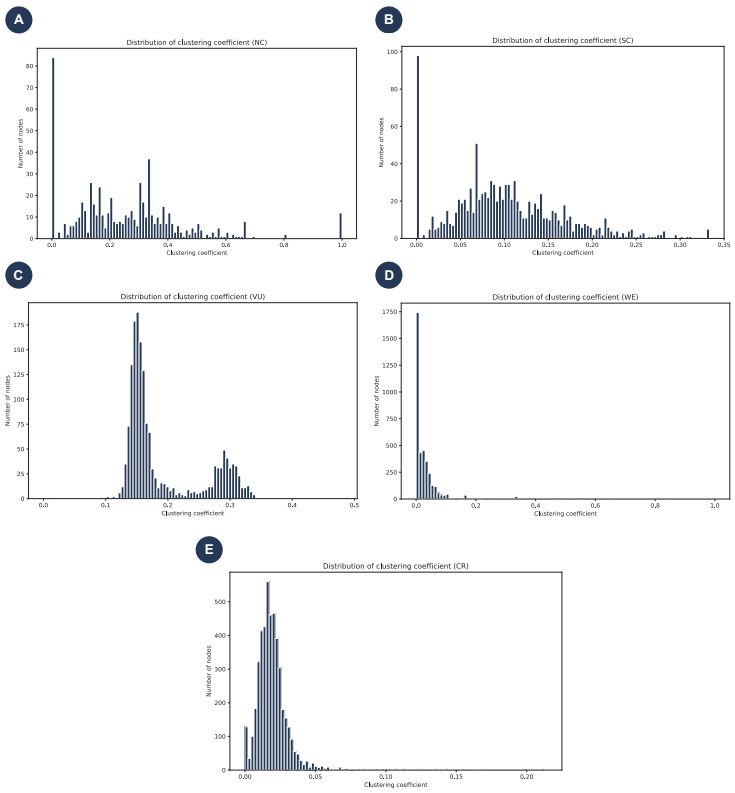

Figure 17: Distribution of clustering coefficients for each node in each dataset: (A) Northern Caucasus populations, (B) Scandinavian populations, (C) Volga-Ural region populations, (D) Western European populations, (E) Eastern Slavs populations (excluding unlabeled vertices).

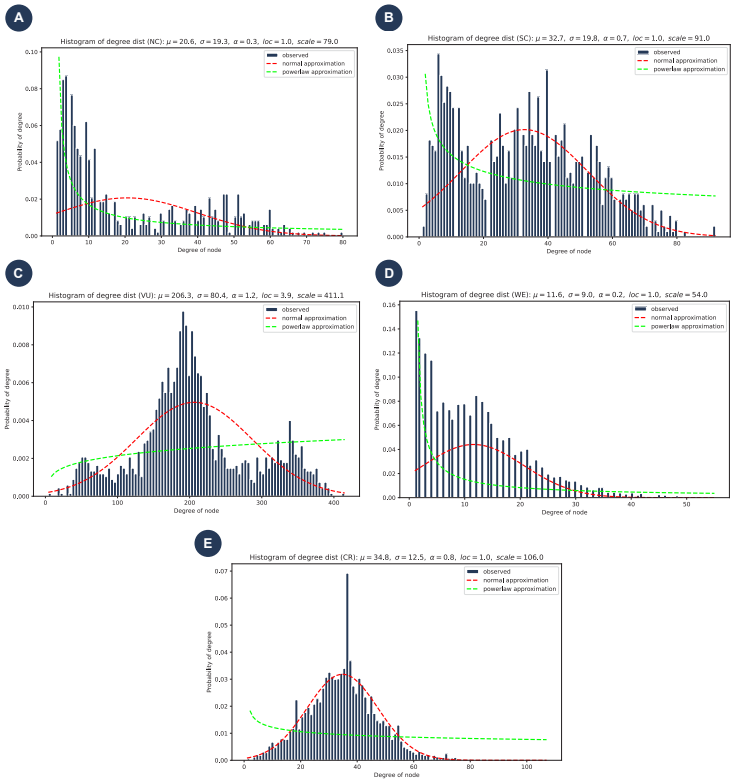

Figure 18: Distribution of degrees of nodes in each dataset: (A) Northern Caucasus populations, (B) Scandinavian populations, (C) Volga-Ural region populations, (D) Western European populations, (E) Eastern Slavs populations (excluding unlabeled vertices). Each distribution was approximated with powerlaw distribution (green line) and normal distrbution (red line).

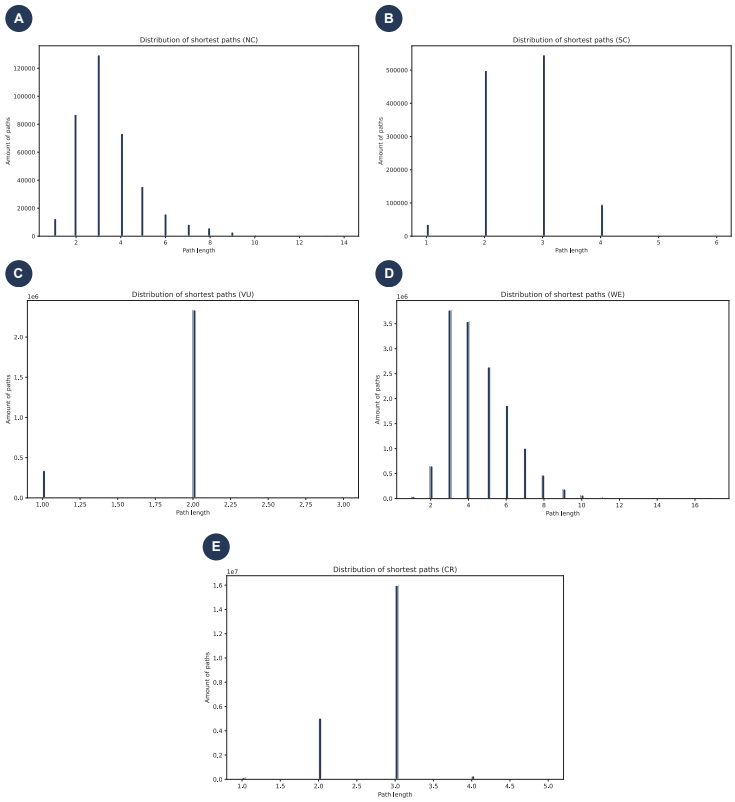

Figure 19: Distribution of all shortest paths in each dataset: (A) Northern Caucasus populations, (B) Scandinavian populations, (C) Volga-Ural region populations, (D) Western European populations, (E) Eastern Slavs populations (excluding unlabeled vertices).

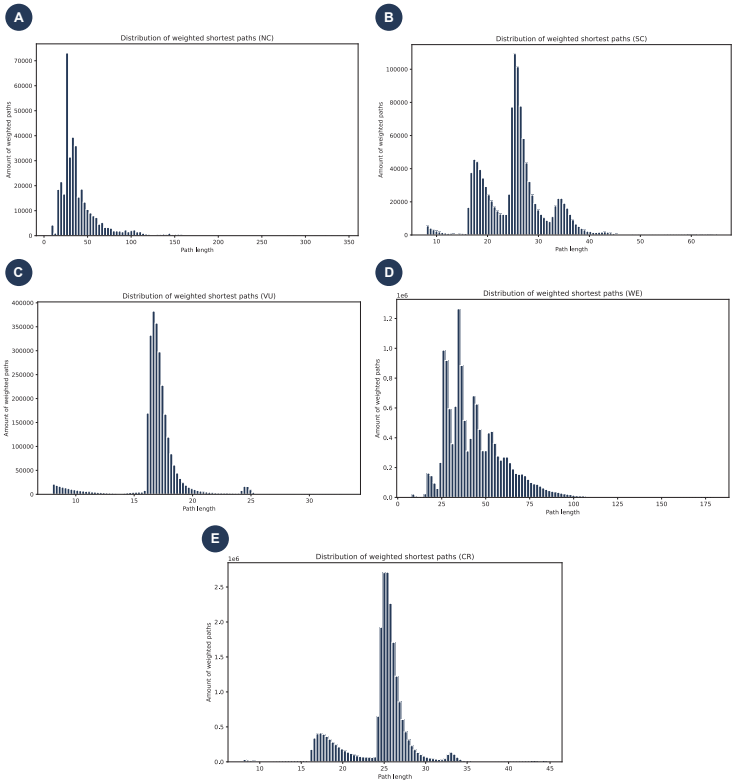

Figure 20: Distribution of weighted shortest paths in each dataset: (A) Northern Caucasus populations, (B) Scandinavian populations, (C) Volga-Ural region populations, (D) Western European populations, (E) Eastern Slavs populations (excluding unlabeled vertices).

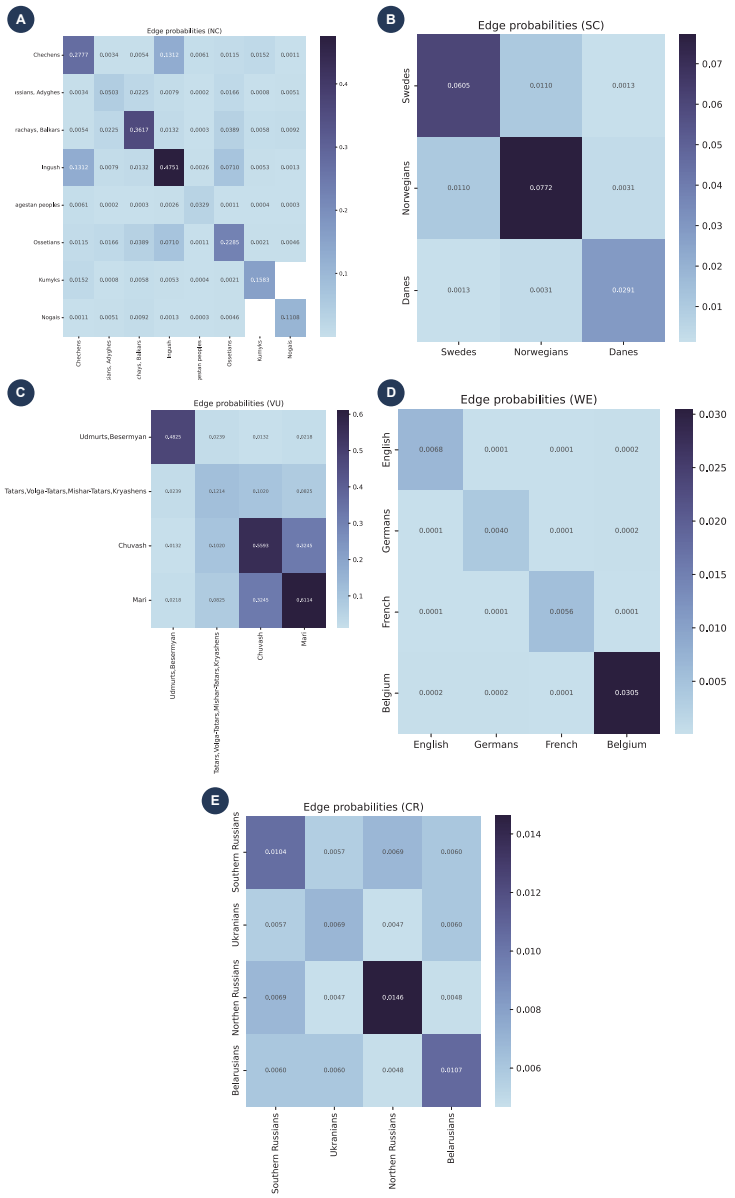

Figure 21: Heatmap of edge probabilities across population classes in each dataset: (A) Northern Caucasus populations, (B) Scandinavian populations, (C) Volga-Ural region populations, (D) Western European populations, (E) Eastern Slavs populations (excluding unlabeled vertices). Each cell in each subfigure represents the probability that node from class i will be connected to node from class j based on real graphs.

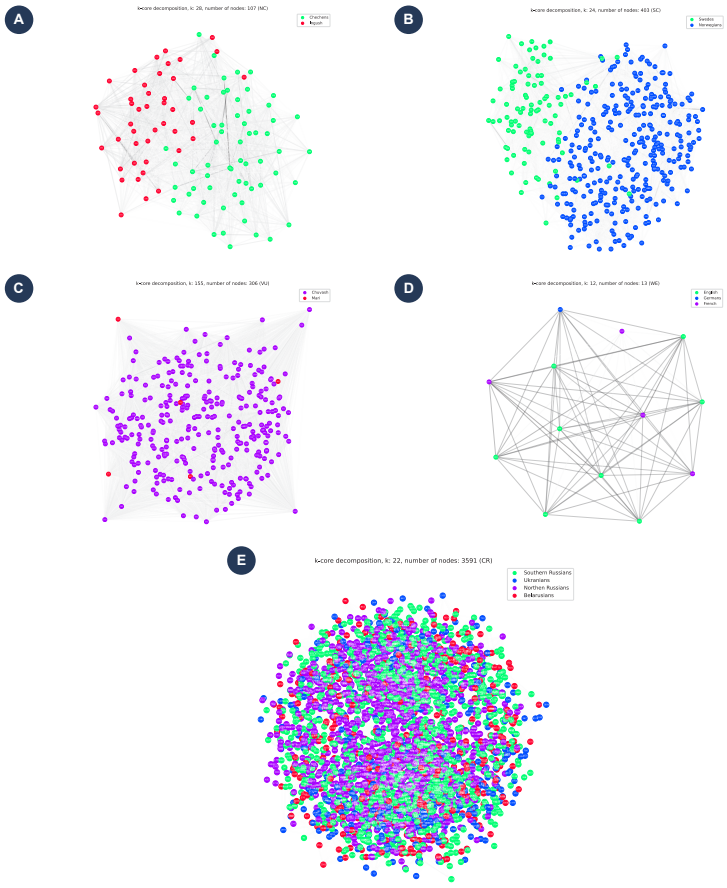

Figure 22: Final result of k-core decomposition in each dataset: (A) Northern Caucasus populations, (B) Scandinavian populations, (C) Volga-Ural region populations, (D) Western European populations, (E) Eastern Slavs populations (excluding unlabeled vertices). Here we showed the subgraphs with maximum available k for each graph and its population composition.

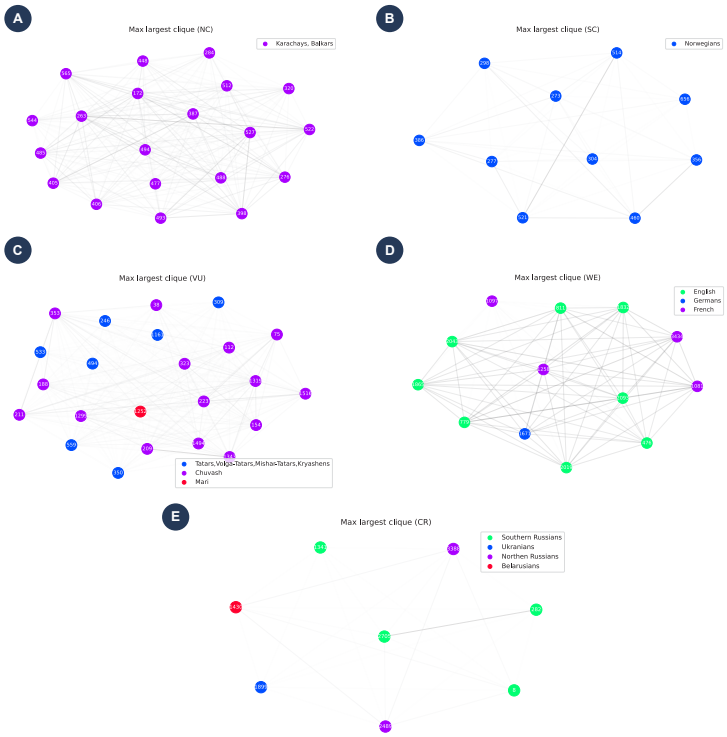

Figure 23: Max largest cliques in each dataset: (A) Northern Caucasus populations, (B) Scandinavian populations, (C) Volga-Ural region populations, (D) Western European populations, (E) Eastern Slavs populations (excluding unlabeled vertices). Here we showed the subgraphs that are largest cliques for each graph and its population composition.

N ACKNOWLEDGEMENTS

This work was supported by a grant for research centers in the field of artificial intelligence provided by the Ministry of Economic Development of the Russian Federation, in accordance with Agreement No. 000000C313925P4E0002 and the agreement with HSE University No. 139-15-2025-009.

O DECLARATION OF LLM USAGE.

Large Language Models (LLMs) were used solely to improve the readability and clarity of the manuscript text. No parts of the analysis, results, or conclusions were generated by LLMs.