Supplementary Information

S1 GC-MERGE model details

S1.1 Model architecture and training

The GC-MERGE architecture is represented in Figure 2. Here, the first layer of the model performs a graph convolution on the initial feature embeddings with an output embedding size of 256, followed by application of ReLU, a non-linear activation function. The second layer of the model performs another graph convolution with the same embedding size of 256 on the transformed representations, again followed by application of ReLU. Next, the output is fed into three successive linear layers of sizes 256, 256, and 2, respectively. A regularization step is performed by using a dropout layer with probability 0.5. The model was trained using ADAM, a stochastic gradient descent algorithm (Kingma and Ba, 2017). We used the PyTorch Geometric package (Fey and Lenssen, 2019) to implement our code. Additional details regarding hyperparameter tuning can be found in the Supplemental Section S1.3.

S1.2 Data set pre-processing

To pre-process the data sets, we used pre-existing pipelines of well known tools. To pre-process the Hi-C data, we used Straw, a software program made available by the Aiden Laboratory (Durand et al., 2016). Interested readers can refer to the tutorial given at the tool's web page: `https://github.com/aidenlab/straw/wiki`. To pre-process the ChIP-seq data sets, we used the BEDTools and BamTools suites (Barnett et al., 2011; Quinlan and Hall, 2010). Readers may find the following tutorial provided by the ENCODE Project to be useful: `https://www.encodeproject.org/chip-seq/histone/`.

S1.3 Hyperparameter tuning

Table S1 details the hyperparameters and the range of values we used to conduct a grid search to determine the optimized model. Specifically, we varied the number of graph convolutional layers, number of linear layers, embedding size for graph convolutional layers, linear layer sizes, and inclusion (or exclusion) of an activation function after the graph convolutional layers. Through earlier iterations of hyperparameter tuning, we also tested the type of activation functions used for the linear layers of the model (ReLU, LeakyReLU, sigmoid, or tanh), methods for accounting for background Hi-C counts, as well as dropout probabilities. Some combinations of hyperparameters were omitted from our grid search because the corresponding model's memory requirements did not fit on the NVIDIA Titan RTX and Quadro RTX GPUs available to us on Brown University's Center for Computation and Visualization (CCV) computing cluster. We recorded the loss curves for the training and validation sets over 800 epochs for the classification task and 1000 epochs for the regression task, by which time the model began to overfit. In addition, the data was split into sets of 70% for training, 15% for validation, and 15% for testing. The optimal hyperparameters for our final model that also proved to be computationally feasible are as follows: 2 graph convolutional layers, 3 linear layers, graph convolutional layer embedding size of 256, linear layer sizes