Supplemental Material

Anchor: Trans-cell Type Prediction of Transcription Factor Binding Sites

Hongyang Li[1], Daniel Quang[1], Yuanfang Guan[1,*]

1. Department of Computational Medicine and Bioinformatics, University of Michigan, 100 Washtenaw Avenue, Ann Arbor, MI 48109, USA

* Corresponding author: gyuanfan@umich.edu

Contents

Extended Methods

Our software Anchor is available on github (https://github.com/GuanLab/Anchor). Anchor has three types of features (DNase-seq; DNA sequence and TF motif; Gencode annotation). Multiple preprocessing steps are required to generate a total of 556 features. A step-by-step instruction is provided below and the example code is also available on github (https://github.com/GuanLab/Anchor/blob/master/DETAILS.md).

Preparing the DNase-seq based features

step1. convert the original read alignment bam to txt file using samtools. e.g.
samtools depth INPUT_BAM > OUTPUT_TXT

step2: for each genomic coordinate, sum the original reads (the output of step1) from all technical or biological replicates, and separate the summed read into 23 file (23 chromosomes: 1-22,X)

step3: to save the computational time and memory, subsample the output of step2 by 1/1000, and rank the subsampled reads from all chromosomes from low to high (for the multi-cell type quantile normalization next)

step4: resize the output of step3 to the same length as the reference cell line (the liver cell line is used in the ENCODE-DREAM challenge)

step5: quantile normalize each cell line (the output of step4) to the reference cell line

step6.1: for each 200bp genomic coordinates with a sliding window of 50bp, calculate the mean, maximum, minimum values using the output of step5 as the “3M-DNase” features; the blacklisted regions (https://sites.google.com/site/anshulkundaje/projects/blacklists) was excluded

step6.2: for each 200bp genomic coordinates with a sliding window of 50bp, calculate the average “3M-DNase” features across all available cell lines using the output of step 6.1 and the average values are further subtracted from each cell line to generate the Δmean feature

step6.3: similar to step6.2, generate the Δmax and Δmin features; the step6.2 and step6.3 generate the “Δ3M-DNase” features, which capture the differences across multiple cell lines and reduces the cell line-specific biases.

step6.4: for each 200bp genomic interval with a sliding window of 50bp, collect the upstream 7 neighboring features and the downstream 7 neighbors from the output of step6.1 to generate a total of 15 neighboring mean features

step6.5: similar to step6.4, generate the 15 neighboring maximum and 15 minimum features. The outputs of step6.4 and step6.5 form the final “3M-DNase-neighbors” features.

step6.6: similar to step6.4, generate the 15 neighboring Δmean features

step6.7: similar to step6.5, generate the 15 neighboring Δmaximum and 15 Δminimum features. The outputs of step6.6 and step6.7 form the final “Δ3M-DNase-neighbors” features.

step7: convert fold-enrichment signal coverage bigWig files to txt files using bigWigToBedGraph. e.g.
bigWigToBedGraph INPUT_BIGWIG OUTPUT_TXT

step8: for each 200bp genomic coordinates with a sliding window of 50bp, count the number of signal occurrence using the output of step7

step9.1: generate the frequency feature by normalizing the output of step8 to range 0-1 through ranking

step9.2: similar to step6.2, generate the Δfrequency feature

step9.3: similar to step6.4, generate the 15 neighboring frequency features

step9.4: similar to step6.4, generate the 15 neighboring Δfrequency features

Preparing the DNA sequence- and TF motif-based features

step10.1: for each TF under consideration, scan the motif Position Weight Matrix (PWM) against human genome sequences to obtain the motif-matching score for each position

step10.2: for each 200bp genomic intervals, select top 3 scores among the 200 positions using the output of step10.1

step11.1: similar to step10.1, convert the genome sequence to its reverse complement. Then for each TF under consideration, scan the motif Position Weight Matrix (PWM) against converted sequences to obtain the motif-matching score for each position

step11.2: similar to step10.2, for each 200bp genomic coordinates, select top 3 scores among the 200 positions using the output of step11.1

step12: select the top 4 out of the 6 scores (3 forward from step10.2 and 3 reverse from step11.2)

step13: for each of the top 4 scores from step12, rank the scores to range 0-1 as the sequence features.

step14: for each of the ranked top 4 scores from step13, generate the 4 upstream and 4 downstream neighboring features, resulting in a total of 32 features = (4 upstream + 4 downstream) * 4 top scores.

Preparing the distance features to the nearest genes

step15: for each genomic coordinate under consideration, calculate the 20 closest distances to a gene using the Gencode annotations (gencode.v19.annotation.gtf downloaded from: https://www.gencodegenes.org/releases/19.html).

Total number of features

In our "Anchor-Final" model, a total of 556 = 120 + 416 +20 features were used:

- 120 DNase-seq based features = 45 (step6.5) + 45 (step6.7) +15 (step9.3) + 15 (step9.4)
- 416 DNA sequence based features = 32 (step14) * 13 TFs (ATF3, CTCF, E2F1, EGR1, FOXA1, FOXA2, GABPA, HNF4A, JUND, NANOG, REST, TAF1, MAX)
- 20 Gencode based features = 20 (step15)

Two examples of "Bound" and "Unbound" 200bp intervals of these 556 features are shown in **Supplemental Fig. 9** and **Supplemental Fig. 10** .

Training XGBoost models

Our Anchor framework was trained on 20 out of 23 chromosomes (Chr 1, Chr 8 and Chr 21 were used as the held-out testing chromosomes) in 43 TF-cell line combinations (**Supplemental Table S2**). The 20 chromosomes were randomly partitioned into setA and setB:

- setA: Chr 2, Chr 4, Chr 6, Chr 7, Chr 12, Chr 13, Chr 15, Chr 16, Chr 17, Chr 20, Chr X
- setB: Chr 3, Chr 5, Chr 9, Chr 10, Chr 11, Chr 14, Chr 18, Chr 19, Chr 22

For each TF, we trained multiple XGBoost models across all the available training cell lines. For example, TAF1 has four training ChIP-seq data in cell line GM12878, H1-hESC, HeLa-S3 and K562. We trained the XGBoost model on the chromosome setA in GM12878 for 1,000 iterations and obtained 1,000 models. To select the best model and avoid overfitting, we evaluated these 1,000 models on the chromosome setB in H1-hESC. The best model was saved as the 1st model and named as "setA-GM12878-setB-H1-hESC". To exploit the information of all the chromosomes and cell lines, we trained and validated (select the best performing model among 1,000 iterations) 4 models in this way:

- "setA-GM12878-setB-H1-hESC"

- “setA-H1-hESC-setB-HeLa-S3”
- “setA-HeLa-S3-setB-K562”
- “setA-K562-setB-GM12878”

Then we switched chromosome setA and setB, and generated another 4 models:

- “setB-GM12878-setA-H1-hESC”
- “setB-H1-hESC-setA-HeLa-S3”
- “setB-HeLa-S3-setA-K562”
- “setB-K562-setA-GM12878”

When we made prediction of TAF1 in a testing cell line, we averaged the predictions from the 8 models described above as the final prediction.

Parameters used to train XGBoost models

```
# choose gbtree or gblinear
booster = gbtree
# choose logistic regression loss function for binary classification
objective = binary:logistic
# Tree Booster Parameters
# step size shrinkage
eta =  0.1
# maximum depth of a tree
max_depth = 7
# the number of round to do boosting
num_round = 1000
# 0 means do not save any model except the final round model
save_period = 10
# The path of training data
data = "train.dat"
test:data="test.dat"
name_pred="output.dat"
```

The source code

For details, please go to Anchor github:
https://github.com/GuanLab/Anchor
https://github.com/GuanLab/Anchor/blob/master/DETAILS.md

(1) ANCHOR.py

```
"""
Script for predicting genome-scale TF binding sites.
Use `ANCHOR.py -h` to see descriptions of the arguments.
"""
import argparse
import os
```

```
def get_args():
      parser = argparse.ArgumentParser(description="ANCHOR - predicting genome-scale TF binding sites.",
       epilog='\n'.join(__doc__.strip().split('\n')[1:]).strip(),
       formatter_class=argparse.RawTextHelpFormatter)
   parser.add_argument('-tf', '--tf', default='TAF1', nargs='+',type=str,
      help='The Trascription Factor to predict (e.g. TAF1)')
   parser.add_argument('-cell', '--cell_line', default='H1-hESC', type=str,
      help='The cell line name of the DNase-seq data')
   args = parser.parse_args()
   return args

def main():
   args = get_args()

   ## 1. Prepare DNase-seq feature
   os.system('python ANCHOR_dnase.py -cell ' + args.cell_line)

   ## 2. Prepare DNA sequence and TF motif features
   os.system('python ANCHOR_sequence.py -tf ' + args.tf)

   ## 3. Prepare gene location features
   os.system('python ANCHOR_gencode.py')

   ## 4. Make predictions
   os.system('python ANCHOR_prediction.py -tf '  + args.tf + \
      ' -cell ' + args.cell_line)

if __name__ == '__main__':
   main()
```

(2) ANCHOR_dnase.py

```
"""
Script for predicting genome-scale TF binding sites.
Use `ANCHOR_dnase.py -h` to see descriptions of the arguments.
"""
import argparse
import os

def get_args():
        parser = argparse.ArgumentParser(description="ANCHOR_dnase - preprocessing DNase-seq data",
```

```
        epilog='\n'.join(__doc__.strip().split('\n')[1:]).strip(),
        formatter_class=argparse.RawTextHelpFormatter)
    parser.add_argument('-i1', '--input1', default='./data/dnase_aln/', type=str,
                help='Directory of the input DNase-seq read alignment BAM files (default: ./data/dnase_aln/)')
    parser.add_argument('-i2', '--input2', default='./data/dnase_fold_coverage/', type=str,
            help='Directory of the input fold-enrichment signal coverage BigWig files (default: ./data/dnase_fold_coverage/)')
    parser.add_argument('-cell', '--cell_line', default='H1-hESC', type=str,
        help='The cell line name of the DNase-seq data')
    parser.add_argument('-o', '--output', default='./data/', type=str,
        help='Directory of the output files (default: ./data/)')
    args = parser.parse_args()
    return args

def main():
    args = get_args()

     # step1. convert bam to txt file; require samtools; # 16 minutes (e.g. for H1-hESC with 2 replicates)
    cmd = 'perl ./preprocess/dnase/transform_bam_to_track.pl ' + \
        args.input1 + ' ' + \
        args.output + 'dnase_track/'
    os.system(cmd)

     # step2: sum raw reads from all tech/bio replicates and separate them into 23 chromosomes; 15-20 hours
    cmd = 'perl ./preprocess/dnase/create_DNase_avg_track_by_chr.pl ' + \
        args.cell_line + ' ' + \
        args.output + 'dnase_track/ ' + \
        args.output + 'dnase_track_avg/'
    os.system(cmd)

     # step3: subsample 1/1000 and rank reads of all chrs for the multi-cell quantile normalization; 12-15 minutes perl cell line
    cmd = 'perl ./preprocess/dnase/approximate_max_min_median.pl ' + \
        args.cell_line + ' ' + \
        args.output + 'dnase_track_avg/ ' + \
        args.output + 'dnase_sort/'
    os.system(cmd)

     # step4: resize to the same length as the reference; require cv2.resize; 1-2 minutes perl cell line
```

```
    cmd = 'python ./preprocess/dnase/resize_dnase_sort.py ' + \
        args.output + 'dnase_sort/ ' + \
        args.output + 'dnase_sort_resize/'
    os.system(cmd)

    # step5: quantile normalization to the reference cell line; 100-150 minutes perl cell line
    # the outputs of step2 and step4 are needed; a reference cell line from step4 is needed
    # by default, we used liver.txt with 3036304 lines as the reference cell line
    cmd = 'perl ./preprocess/dnase/normalize_by_anchor.pl ' + \
        args.cell_line + ' ' + \
        args.output + 'dnase_track_avg/ ' + \
        args.output + 'dnase_sort_resize/ ' + \
        args.output + 'dnase_track_avg_anchor/ ' + \
        './data/ref/liver.txt'
    os.system(cmd)

    # step6.1: generate the mean, max, min features; 200 minutes perl cell line
    # a reference file the genomic coordinates excluding blacklisted regions is needed; this file looks like this:
    # chr1  600 800
    # chr1  650 850
    # chr1  700 900
    # the blacklist can downloaded from here:
    # https://sites.google.com/site/anshulkundaje/projects/blacklists
    cmd = 'perl ./preprocess/dnase/create_DNase_mmm.pl ' + \
        args.cell_line + ' ' + \
        args.output + 'dnase_track_avg_anchor/ ' + \
        args.output + 'anchor_bam_dnase/ ' + \
        args.output + 'anchor_bam_dnase_max_min/ ' + \
        './data/ref/test_regions.blacklistfiltered.bed'
    os.system(cmd)

    # step6.2: generate the delta- mean feature; 40 minutes for the DNase-seq data of 13 cell lines
    # prepare the anchor_bam_dnase from multiple cell line first - the step calculate the differences across cell lines
    cmd = 'perl ./preprocess/dnase/create_DNase_diff.pl ' + \
        args.output + 'anchor_bam_dnase/ ' + \
        args.output + 'anchor_bam_dnase_diff/'
    os.system(cmd)

    # step6.3: generate the delta- max, min features; 100 minutes for the DNase-seq data of 13 cell lines
```

```
    # prepare the anchor_bam_dnase_max_min from multiple cell line first - the step calculate the differences across cell lines
    cmd = 'perl ./preprocess/dnase/create_DNase_max_min_diff.pl ' + \
        args.output + 'anchor_bam_dnase_max_min/ ' + \
        args.output + 'anchor_bam_dnase_max_min_diff/'
    os.system(cmd)

    # step6.4: generate the 15 neighboring mean features; 30 minutes per cell line
    cmd = 'perl ./preprocess/dnase/create_DNase_largespace.pl ' + \
        args.output + 'anchor_bam_dnase/ ' + \
        args.output + 'anchor_bam_dnase_largespace/'
    os.system(cmd)

    # step6.5: generate the 30 neighboring max and min features; 60 minutes per cell line
    cmd = 'perl ./preprocess/dnase/create_DNase_max_min_largespace.pl ' + \
        args.output + 'anchor_bam_dnase_max_min/ ' + \
        args.output + 'anchor_bam_dnase_max_min_largespace/'
    os.system(cmd)

    # step6.6: generate the 15 neighboring delta-mean features; 30 minutes per cell line
    cmd = 'perl ./preprocess/dnase/create_DNase_largespace.pl ' + \
        args.output + 'anchor_bam_dnase_diff/ ' + \
        args.output + 'anchor_bam_dnase_diff_largespace/'
    os.system(cmd)

    # step6.7: generate the 30 neighboring delta-max and delta-min features; 60 minutes per cell line
    cmd = 'perl ./preprocess/dnase/create_DNase_max_min_largespace.pl ' + \
        args.output + 'anchor_bam_dnase_max_min_diff/ ' + \
        args.output + 'anchor_bam_dnase_max_min_diff_largespace/'
    os.system(cmd)

    ## here fold-enrichment frequency feature ##

    # step7: convert fold-enrichment bigWig to txt; 2 minutes per cell line
    cmd = 'perl ./preprocess/dnase/transform_bigwig_to_txt.pl ' + \
        args.input2 + ' ' + \
        args.output + 'dnase_fold_coverage_big/'
    os.system(cmd)

   # step8: count the number of signal occurrence and ignore values to generate frequency features; 60 minutes per cell line
```

```
    # the reference file of the genomic coordinates excluding blacklisted regions is needed
    # the blacklist can downloaded from here:
    # https://sites.google.com/site/anshulkundaje/projects/blacklists
    cmd = 'perl ./preprocess/dnase/produce_frequency.pl ' + \
        args.cell_line + ' ' + \
        args.output + 'dnase_fold_coverage_big/ ' + \
        args.output + 'frequency/ ' + \
        './data/ref/test_regions.blacklistfiltered.bed'
    os.system(cmd)

    # step9.1: generate the frequency feature by normalizing the counting number by ranking; 2 minutes per cell line
    cmd = 'perl ./preprocess/dnase/produce_frequency_rank.pl ' + \
        args.output + 'frequency/ ' + \
        args.output + 'frequency_rank/'
    os.system(cmd)

    # step9.2: generate the delta-frequency feature; 50 minutes for the DNase-seq data of 13 cell lines
    cmd = 'perl ./preprocess/dnase/create_frequency_diff.pl ' + \
        args.output + 'frequency_rank/ ' + \
        args.output + 'frequency_rank_diff/'
    os.system(cmd)

    # step9.3: generate the 15 neighboring frequency features; 50 minutes per cell line
    cmd = 'perl ./preprocess/dnase/create_frequency_largespace.pl ' + \
        args.output + 'frequency_rank/ ' + \
        args.output + 'frequency_rank_largespace/'
    os.system(cmd)

    # step9.4: generate the 15 neighboring delta- frequency features; 50 minutes per cell line
    cmd = 'perl ./preprocess/dnase/create_frequency_largespace.pl ' + \
        args.output + 'frequency_rank_diff/ ' + \
        args.output + 'frequency_rank_diff_largespace/'
    os.system(cmd)

if __name__ == '__main__':
    main()
```

(3) ANCHOR_sequence.py

```
"""
Script for predicting genome-scale TF binding sites.
Use `ANCHOR_sequence.py -h` to see descriptions of the arguments.
```

```
"""
import argparse
import os

def get_args():
        parser = argparse.ArgumentParser(description="ANCHOR_sequence - prepare DNA sequence based features",
      epilog='\n'.join(__doc__.strip().split('\n')[1:]).strip(),
      formatter_class=argparse.RawTextHelpFormatter)
   parser.add_argument('-tf', '--tf', default='TAF1', nargs='+',type=str,
      help='TFs under consideration (e.g. TAF1 MAX')
   parser.add_argument('-motif', '--motif_dir', default='./data/motif/', type=str,
      help='Directory of the sequence motifs (e.g.: ./data/motif/TAF1)')
   parser.add_argument('-seq', '--seq_dir', default='./data/hg_genome/', type=str,
      help='Directory of the human genome sequences (e.g.: ./data/hg_genome/chr1)')
   parser.add_argument('-o', '--output', default='./data/', type=str,
      help='Directory of the output files (default: ./data/)')
   args = parser.parse_args()
   return args

def main():
   args = get_args()

   all_chr=['chr1','chr2','chr3','chr4','chr5','chr6','chr7','chr8','chr9','chr10', \
      'chr11','chr12','chr13','chr14','chr15','chr16','chr17','chr18','chr19','chr20', \
      'chr21','chr22','chrX']

    # step10.1: scan TF-motif PWMs against DNA sequences to obtain score for each position; 60-120 minutes per TF per chromosome (23 chrs!)
   for the_tf in args.tf:
      print(the_tf)
      for the_chr in all_chr:
         cmd = 'python ./preprocess/sequence/scan_motif_by_chr.py ' + \
            the_chr + ' ' + \
            the_tf + ' ' + \
            args.motif_dir + ' ' + \
            args.seq_dir + ' ' + \
            args.output + 'seq_forward/'
         os.system(cmd)

   # step10.2: select top 3 scores over forward 200bp sequences; 150 minutes per cell line
   for the_tf in args.tf:
      cmd = 'perl ./preprocess/sequence/create_motif_top3_ru_forward.pl ' + \
```

```
            the_tf + ' ' + \
            args.output + 'seq_forward/ ' + \
            args.output + 'seq_forward_top3/'
        os.system(cmd)

    # step11.1: scan TF-motif PWMs against the reverse-complement DNA sequences; 60-120 minutes per TF per chr
    for the_tf in args.tf:
        print(the_tf)
        for the_chr in all_chr:
            cmd = 'python ./preprocess/sequence/scan_motif_by_chr_reverse.py ' + \
                the_chr + ' ' + \
                the_tf + ' ' + \
                args.motif_dir + ' ' + \
                args.seq_dir + ' ' + \
                args.output + 'seq_reverse/'
            os.system(cmd)

    # step11.2: select top 3 scores over reverse 200bp sequences; 150 minutes per cell line
    for the_tf in args.tf:
        cmd = 'perl ./preprocess/sequence/create_motif_top3_ru_forward.pl ' + \
            the_tf + ' ' + \
            args.output + 'seq_reverse/ ' + \
            args.output + 'seq_reverse_top3/'
        os.system(cmd)

    # step12: select the top 4 out of the top 6 = 3forward + 3reverse scores; 10 minutes per TF
    for the_tf in args.tf:
        cmd = 'perl ./preprocess/sequence/find_max_forward_reverse_top_4.pl ' + \
            args.output + 'seq_forward_top3/ ' + \
            args.output + 'seq_reverse_top3/ ' + \
            args.output + 'seq_top4/'
        os.system(cmd)

    # step13: rank the features to 0-1; 10 minutes per TF
    for the_tf in args.tf:
        cmd = 'perl ./preprocess/sequence/create_top4_rank_ru.pl ' + \
            args.output + 'seq_top4/ ' + \
            args.output + 'seq_top4_rank/'
        os.system(cmd)

    # step14: generate the 32 neighboring ranked features; 80 minutes per TF
    for the_tf in args.tf:
```

```
        cmd = 'perl ./preprocess/sequence/create_largespace_tomax_top4_rank.pl ' + \
            args.output + 'seq_top4_rank/ ' + \
            args.output + 'seq_top4_rank_largespace/'
        os.system(cmd)

if __name__ == '__main__':
    main()
```

(4) ANCHOR_gencode.py

```
"""
Script for predicting genome-scale TF binding sites.
Use `ANCHOR_gencode.py -h` to see descriptions of the arguments.
"""
import argparse
import os

def get_args():
        parser = argparse.ArgumentParser(description="ANCHOR_gencode - preprocessing the
gencode data",
        epilog='\n'.join(__doc__.strip().split('\n')[1:]).strip(),
        formatter_class=argparse.RawTextHelpFormatter)
    parser.add_argument('-o', '--output', default='./data/top_20', type=str,
        help='The output file (default: ./data/top_20)')
    args = parser.parse_args()
    return args

def main():
    args = get_args()

    # step15: for each genomic coordinate, calculate the 20 closest distances to a gene; 8 hours
    # 1.the gencode (e.g. gencode.v19.annotation.gtf) is required
    # (downloaded from: https://www.gencodegenes.org/releases/19.html)
    # 2.the reference file the genomic coordinates excluding blacklisted regions is needed
      # it returns a file 'top_20' - the top 20 closes distances to a gene of each coordinate under
consideration
    cmd = 'perl ./preprocess/gencode/create_gene_distance_top20_uniq.pl ' + \
        args.output + ' ' + \
        './data/ref/gencode.v19.annotation.gtf ' + \
        './data/ref/test_regions.blacklistfiltered.bed'
    os.system(cmd)

if __name__ == '__main__':
```

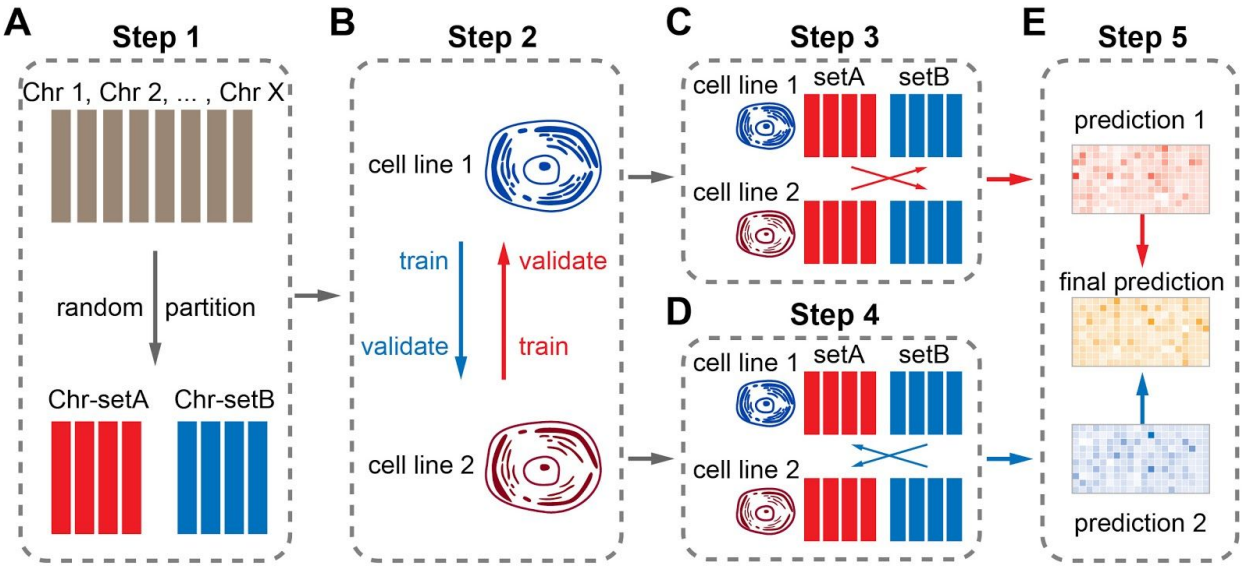
```
    main()
```

(5) ANCHOR_prediction.py

```
"""
Script for predicting drug synergy.
Use `ANCHOR_prediction.py -h` to see descriptions of the arguments.
"""
import argparse
import os

def get_args():
    parser = argparse.ArgumentParser(description="ANCHOR - prediction",
        epilog='\n'.join(__doc__.strip().split('\n')[1:]).strip(),
        formatter_class=argparse.RawTextHelpFormatter)
    parser.add_argument('-tf', '--tf', default='TAF1', type=str,
        help='The TF name (e.g. TAF1)')
    parser.add_argument('-cell', '--cell_line', default='H1-hESC', type=str,
        help='The cell line name (e.g. H1-hESC)')
    parser.add_argument('-i', '--input_dir', default='./data/', type=str,
        help='Directory of the features (default: ./data/)')
    parser.add_argument('-m', '--mode', default='fast', type=str,
        help='The prediction mode (fast: only use single TF sequence motif features; full: use all 13
TF sequence motif features (warning: need more prediction time!)')
    args = parser.parse_args()
    return args

def main():
    args = get_args()

    ## F: single-tf sequence motif feature; without delta-frequency feature
    cmd = 'perl ./prediction/anchor/F/prediction_F.pl ' + \
        args.tf + ' ' + \
        args.cell_line + ' ' + \
        args.input_dir
    os.system(cmd)
    os.system('perl ./prediction/anchor/F/concatenate_prediction.pl')

    ## G: single-tf sequence motif feature; with delta-frequency feature
    cmd = 'perl ./prediction/anchor/G/prediction_G.pl ' + \
        args.tf + ' ' + \
        args.cell_line + ' ' + \
        args.input_dir
    os.system(cmd)
```

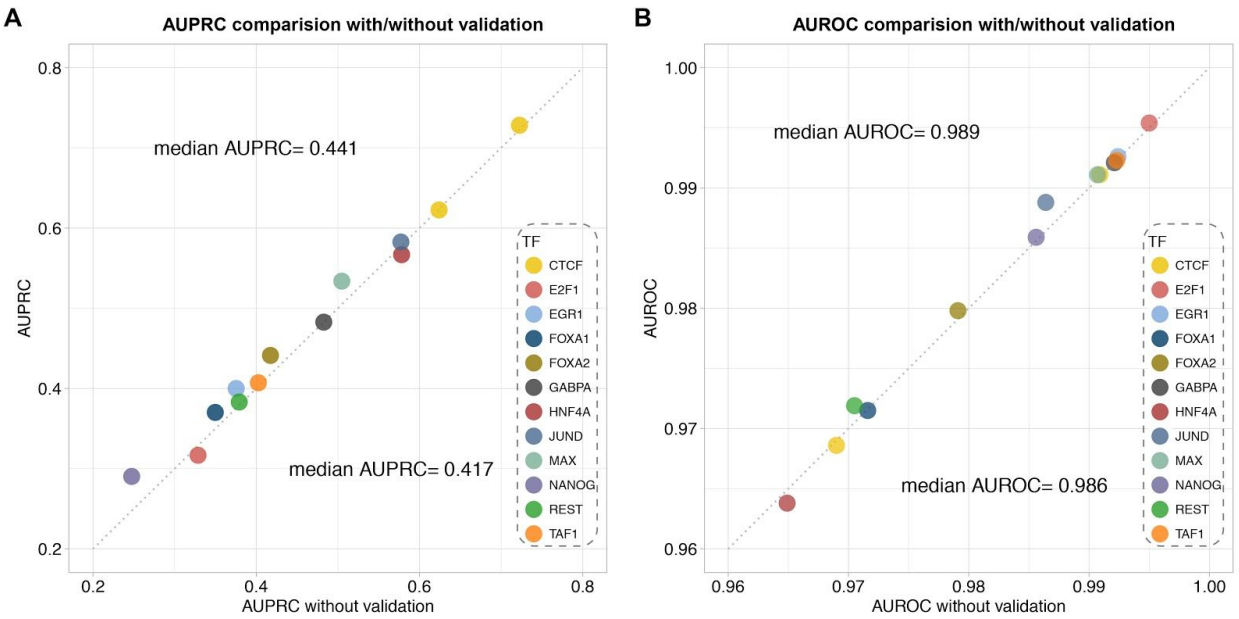
```
    os.system('perl ./prediction/anchor/G/concatenate_prediction.pl')

    if args.mode == 'full':
        print("Full ANCHOR prediction mode (need more time):")

        ## H: multi-tf sequence motif feature; without delta-frequency feature
        print("Run model H")
        cmd = 'perl ./prediction/anchor/H/prediction_H.pl ' + \
            args.tf + ' ' + \
            args.cell_line + ' ' + \
            args.input_dir
        os.system(cmd)
        os.system('perl ./prediction/anchor/H/concatenate_prediction.pl')

        ## I: multi-tf sequence motif feature; with delta-frequency feature
        print("Run model I")
        cmd = 'perl ./prediction/anchor/I/prediction_I.pl ' + \
            args.tf + ' ' + \
            args.cell_line + ' ' + \
            args.input_dir
        os.system(cmd)
        os.system('perl ./prediction/anchor/I/concatenate_prediction.pl')

        print("Ensemble")
        os.system('perl ./prediction/anchor/ensemble_full.pl')

    else:
        print("Ensemble")
        os.system('perl ./prediction/anchor/ensemble_single_tf.pl')

if __name__ == '__main__':
    main()
```

Supplemental Figures



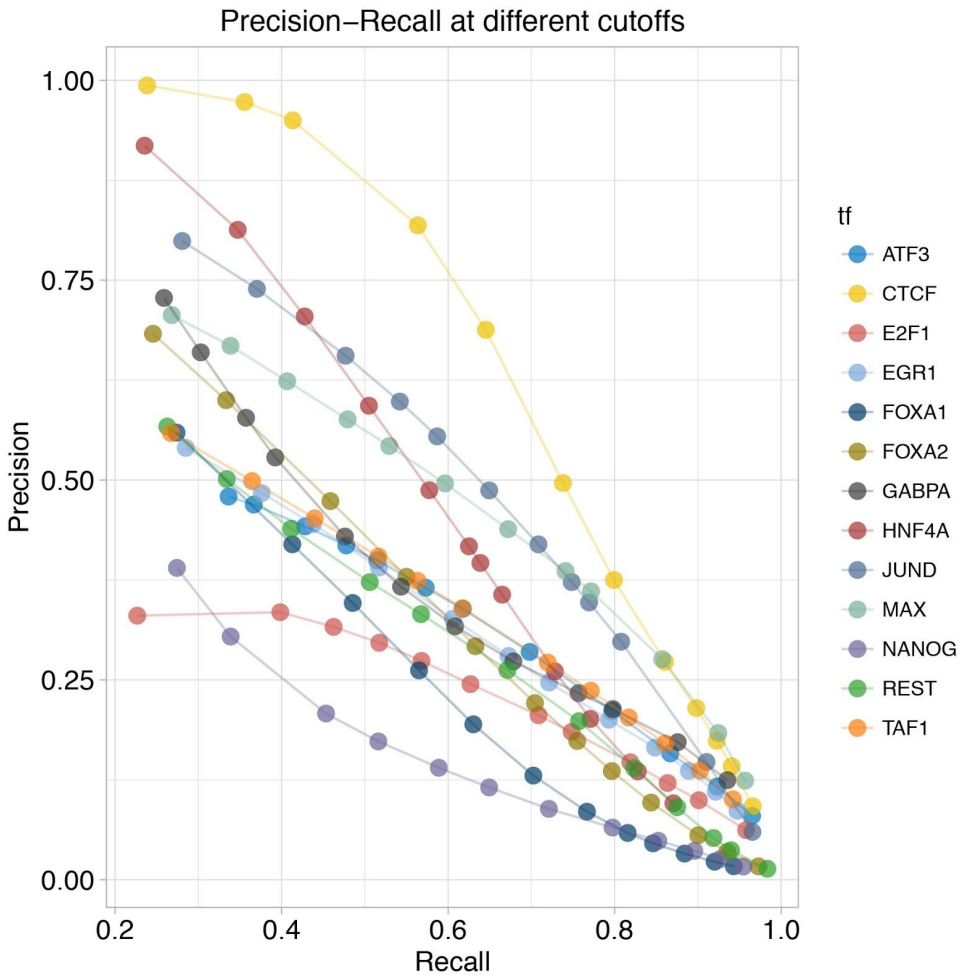
Supplemental Fig. S1. The five major steps of the crisscross training-validation strategy in Anchor.

A. We randomly divide the training chromosomes into two fixed sets, A and B, as the training and validation sets. **B.** We train models on one cell type with the training chromosomes and validate models against another cell type with the validation chromosomes. **C.** Train the XGBoost model on chromosome set A in cell type 1 with 1,000 iterations, and validate on set B in cell type 2 (setA-cell1-setB-cell2). Then we train and validate on setA-cell2-setB-cell1 in the crisscross fashion to avoid overfitting. **D.** Similar to step 3 but switch the train and validation chromosome sets. **E.** Average the predictions of all the models in step 3 and 4 as our final predictions to incorporate information from multiple cell types.

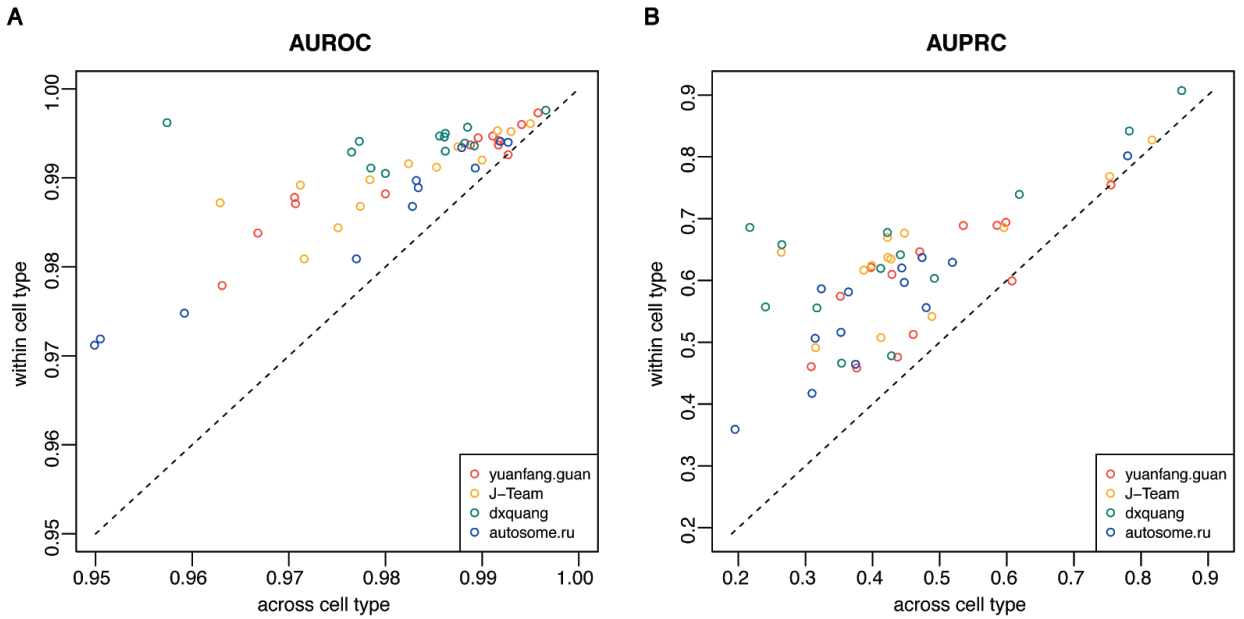


Supplemental Fig. S2. Prediction performances of Anchor model with and without validation

In our Anchor framework, a XGBoost model was trained for 1,000 iterations and the validation set was used to select the best performing iteration and save the best model (y-axis). The model without validation was saved in the last training iteration (x-axis), which was overfitted to the training data. The prediction **A.** AUPRCs and **B.** AUROCs on the 13 testing TF-cell type pairs were calculated with TFs shown in different colors.

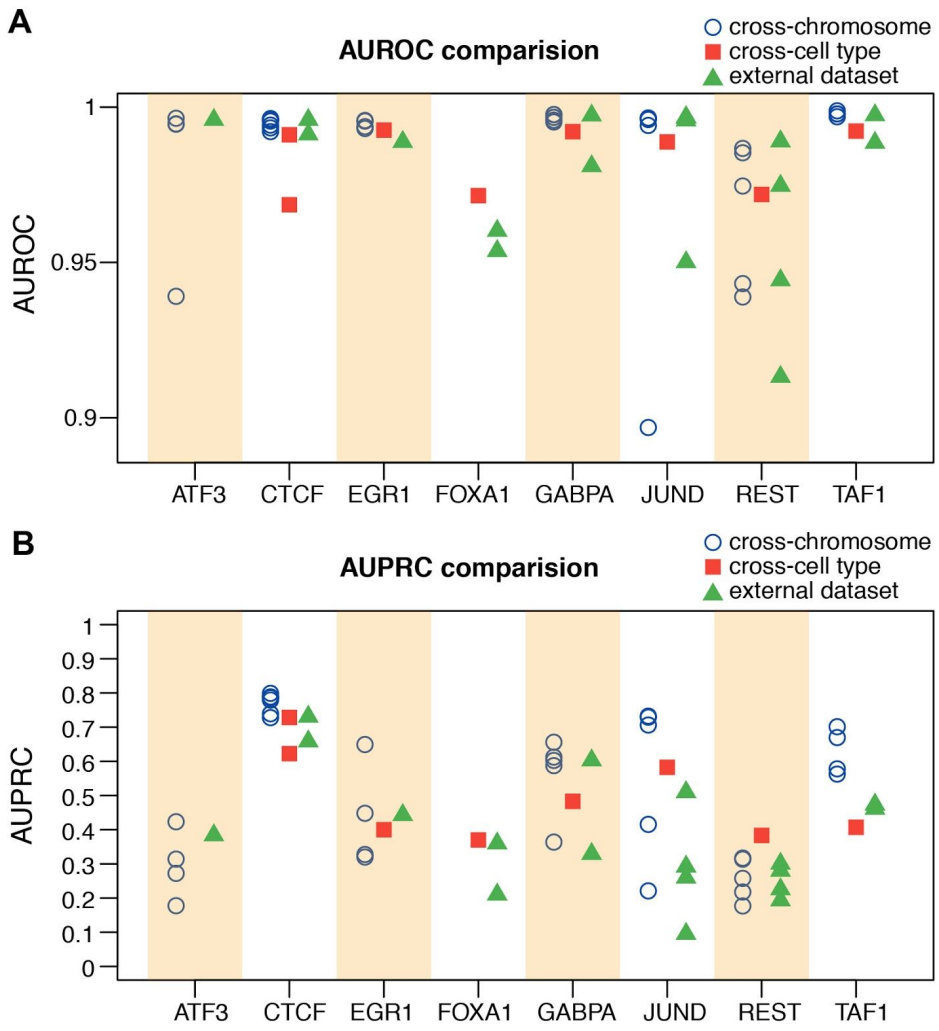


Supplemental Fig. S3. The precision and recall values at different cutoffs of 13 TFs. For each TF, 12 cutoffs were used to threshold the original predictions - the predictions above the cutoff were treated as binding, otherwise non-binding. The 12 corresponding precision (y-axis) and recall (x-axis) pairs were shown as dots. The complete values to generate this plot is shown in **Supplemental Table S3**.



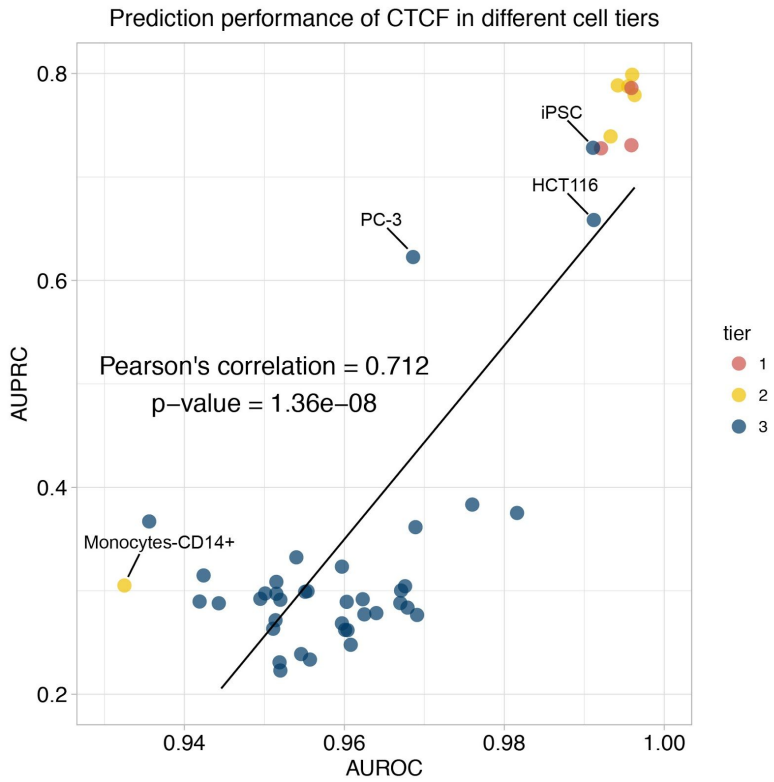
Supplemental Fig. S4. The within-cell and cross-cell predictive performances of the top 4 teams in the ENCODE-DREAM challenge

The **A**. AUROC and **B**. AUPRC comparison of within-cell and cross-cell predictions for the 13 testing TF-cell pairs (**Supplemental Table S5**). The top 4 teams are shown in different colors. The within-cell predictions achieve higher AUROCs and AUPRCs than the cross-cell predictions.

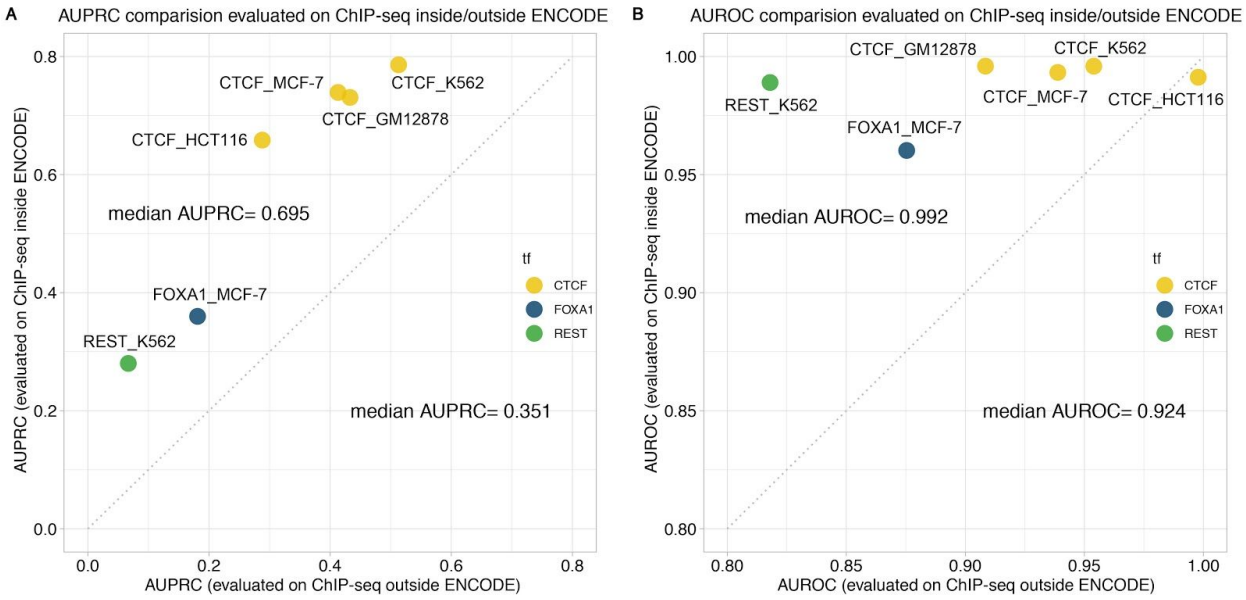


Supplemental Fig. S5. Prediction performances on ChIP-seq data within and outside the ENCODE-DREAM challenge.

The prediction **A.** AUROCs and **B.** AUPRCs on 18 ChIP-seq data outside the ENCODE-DREAM challenge are shown as green triangles. For comparison, the performances for these 8 TFs on the ENCODE-DREAM data are shown as blue circles and red squares, which are the same results shown in **Fig 2E,F**. The performances are comparable on ChIP-seq data inside and outside the challenge. Of note, we performed a total of 55 predictions and the results for the rest 37 ones are CTCF in tier-3 cell lines shown in **Supplemental Fig S6.** The complete values for all the 55 predictions can be found in **Supplemental Table S6.**



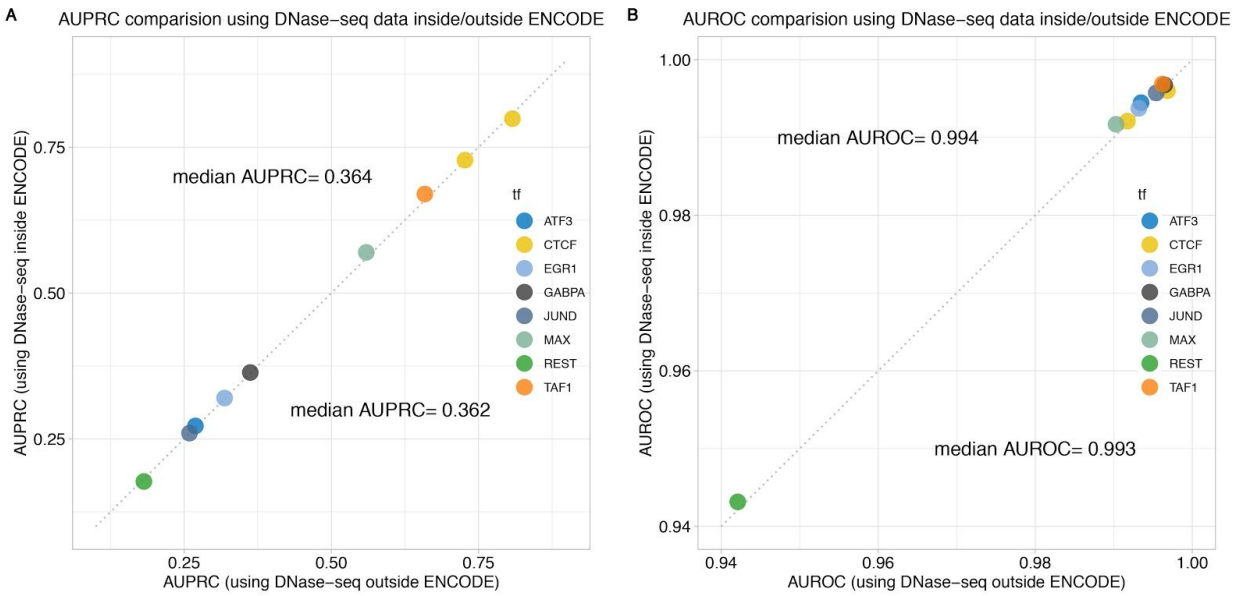
Supplemental Fig. S6. The prediction performances of CTCF in multiple cell lines.
The AUROCs (x-axis) and AUPRCs (y-axis) of CTCF in tier 1, 2 and 3 cell line are shown in different colors. In general, the performances in tier 1 and 2 cell lines are higher than the performances in tier 3 cell lines, of which the DNase-seq has relatively low-coverage. The complete values can be found in **Supplemental Table S6.**



Supplemental Fig. S7. Performance comparison on ChIP-seq data within and outside the ENCODE project.

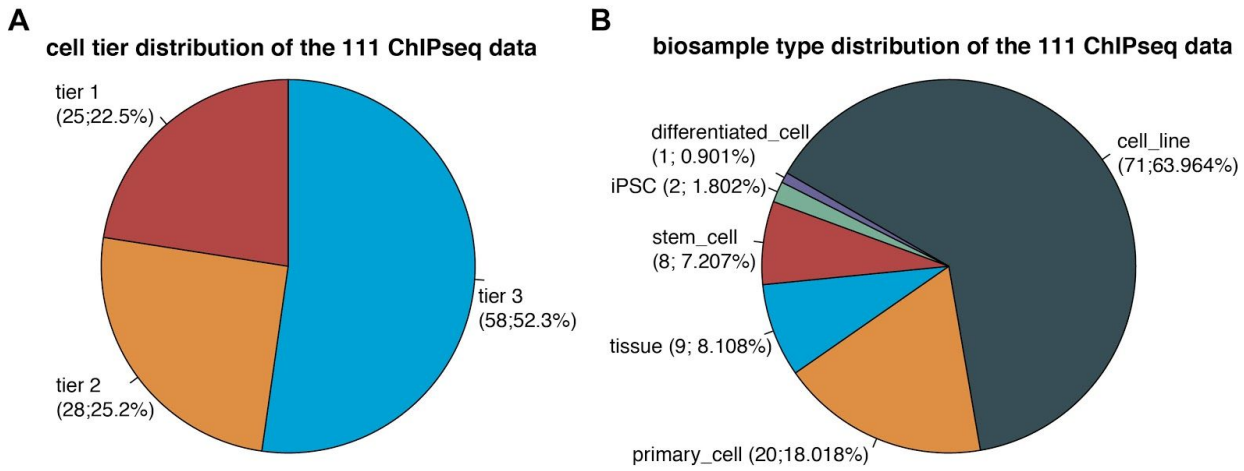
We evaluated the **A**. AUPRC and **B**. AUROC of Anchor prediction on ChIP-seq data within and outside the ENCODE project. The six ChIP-seq data outside the ENCODE project were downloaded from

1. CTCF-GM12878 (https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM1233887)
2. CTCF-HCT116 (https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM1224649)
3. CTCF-K562 (https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE70764)
4. CTCF-MCF-7 (https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM2067524)
5. FOXA1-MCF-7 (https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM2257822)
6. REST-K562 (https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM831015)



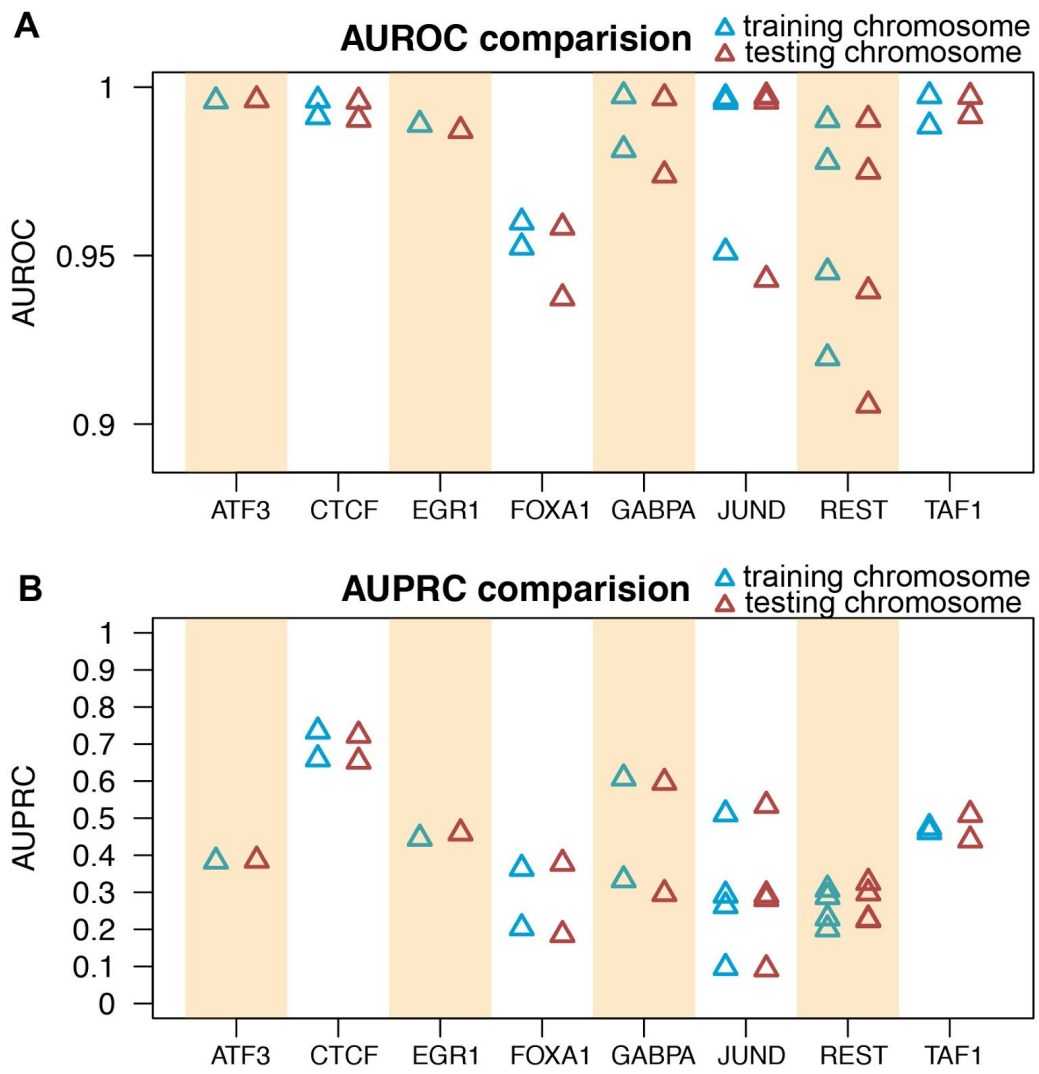
Supplemental Fig. S8. The prediction performances using DNase-seq data within and outside ENCODE.

The **A**. AUPRCs and **B**. AUROCs of predictions on 9 TF-cell type pairs using DNase-seq data within ENCODE (y-axis) and outside ENCODE (x-axis; Roadmap). The complete values can be found in **Supplemental Table S7.**



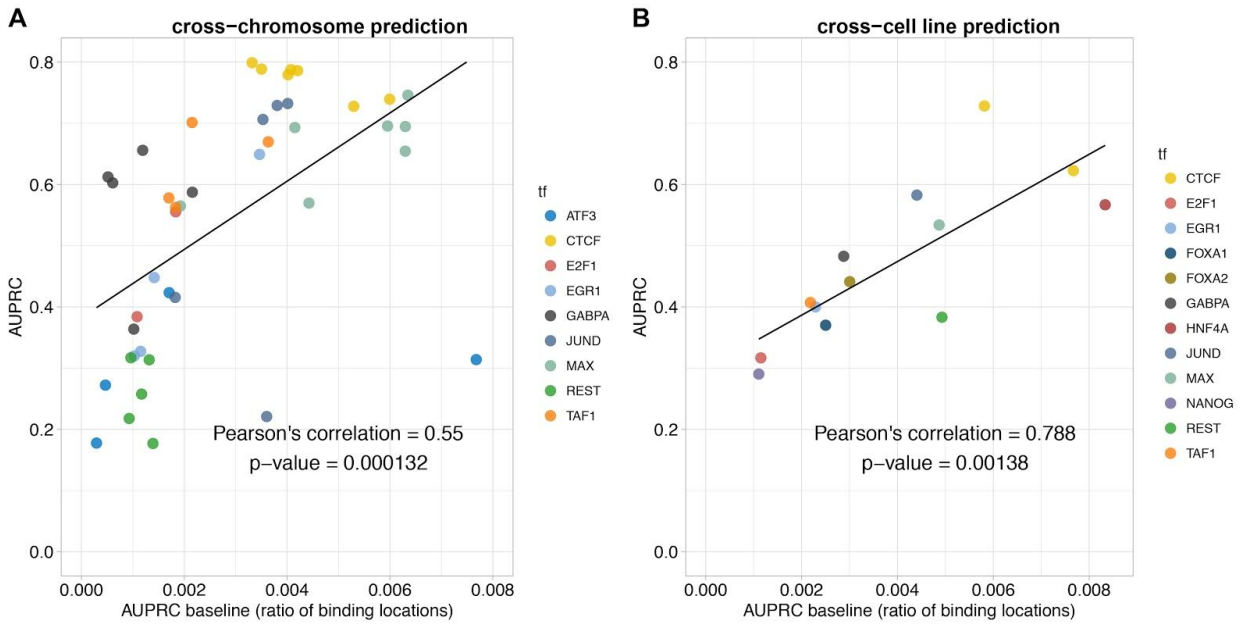
Supplemental Fig. S9. The distributions of cell tier and biosample type of the 111 ChIP-seq data used in this work.

We have summarized the **A**. cell tier and **B**. biosample type distributions of the 111 = 56 + 55 ChIP-seq data investigated in this work. Of note, 56 of them were used in the ENCODE-DREAM challenge and the other 55 outside challenge were downloaded from the ENCODE project data portal. The accession numbers for the 55 ChIP-seq data are provided in **Supplemental Table S6**.



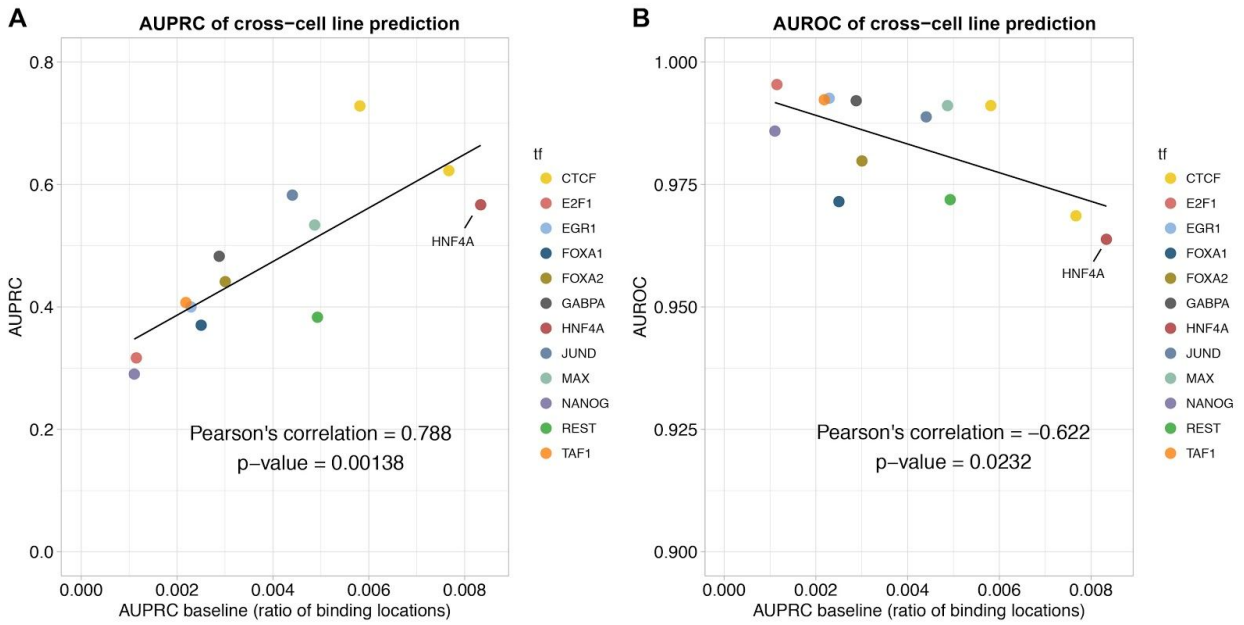
Supplemental Fig. S10. Prediction performance comparison between the 20 training and 3 testing chromosomes.

We evaluated the **A**. AUPRC and **B**. AUROC of our model on all 23 chromosomes in the 18 TF-cell type pairs outside the ENCODE-DREAM challenge (blue - the 20 training chromosomes; red - the 3 held-out testing chromosomes). The difference in performance for the 20 training chromosomes versus the 3 testing chromosomes is almost negligible.



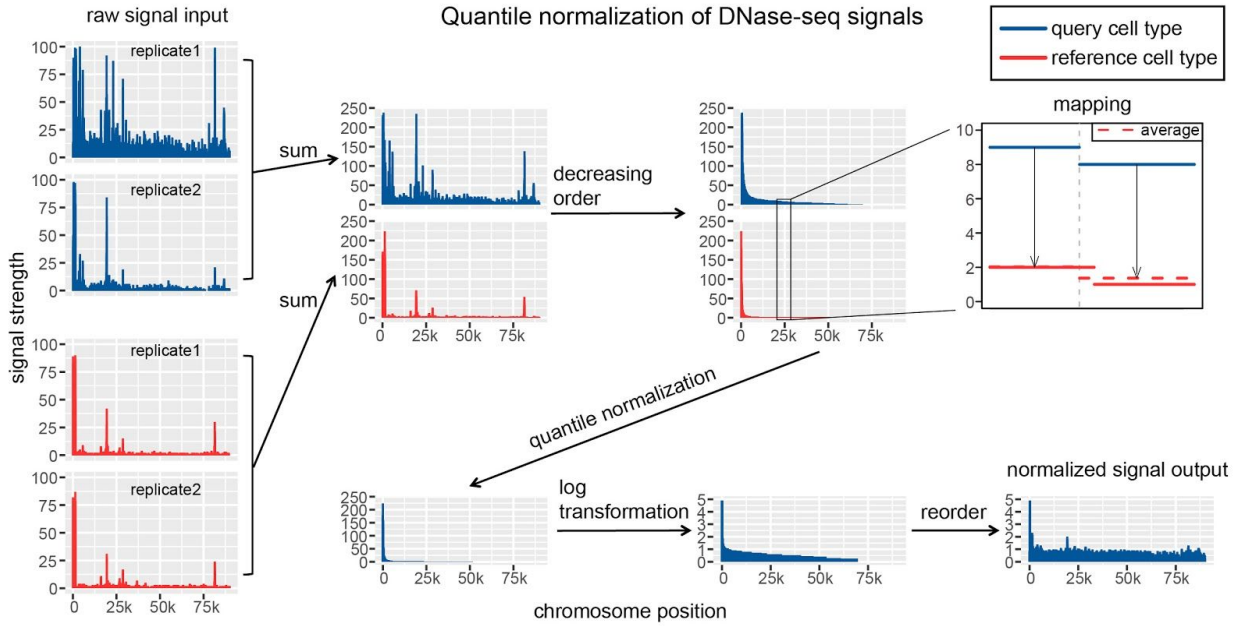
Supplemental Fig. S11. The relationship between prediction AUPRCs and the corresponding AUPRC baselines of random prediction.

The AUPRC baseline corresponds to the ratio of observed binding intervals over all intervals under consideration across the genome. We have plotted the AUPRC of **A**. cross-chromosome and **B**. cross-cell line predictions versus the baseline for the 43 training ChIP-seq data in the ENCODE-DREAM challenge. When evaluating the predictions across chromosomes (corresponding to the blue circles in **Fig. 2F**), we observed a significant medium correlation = 0.55 between the baseline and the AUPRCs. When evaluating the performance across cell types (corresponding to the red square in **Fig. 2F**), we observed a significant strong correlation = 0.788.



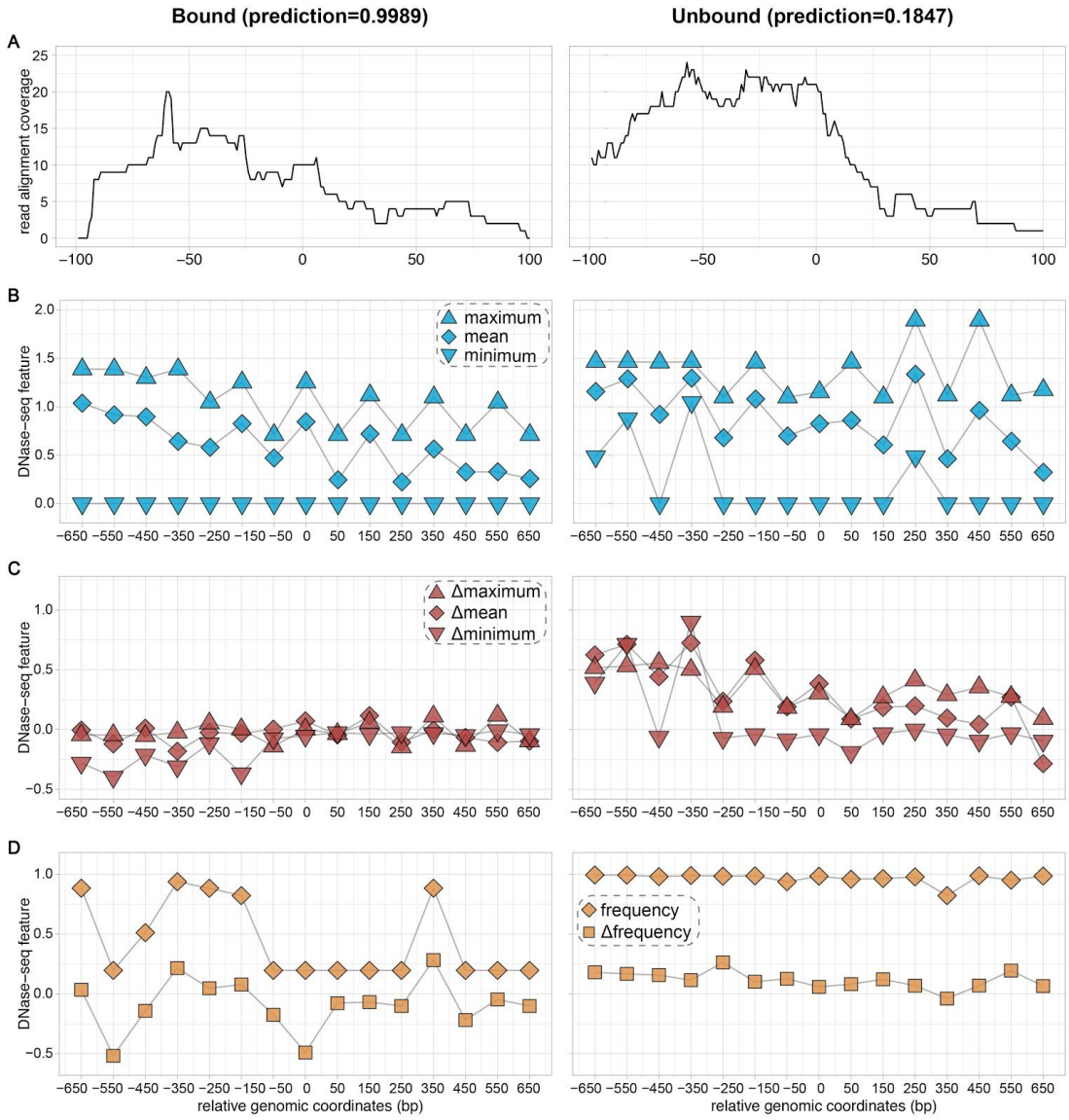
Supplemental Fig. S12. The comparison between prediction performances and the AUPRC baseline.

The AUPRC baseline corresponds to the ratio of observed binding intervals over all intervals under consideration across the genome. We have plotted the **A**. AUPRC and **B**. AUROC versus the baseline for the 13 final test ChIP-seq data in the ENCODE-DREAM challenge. Of note, the HNF4A is highlighted in dark red at top-right. It has the highest AUPRC baseline = 0.00833, resulting in its relatively high prediction AUPRC. There is a significantly strong positive correlation (0.788) between AUPRC and the baseline.

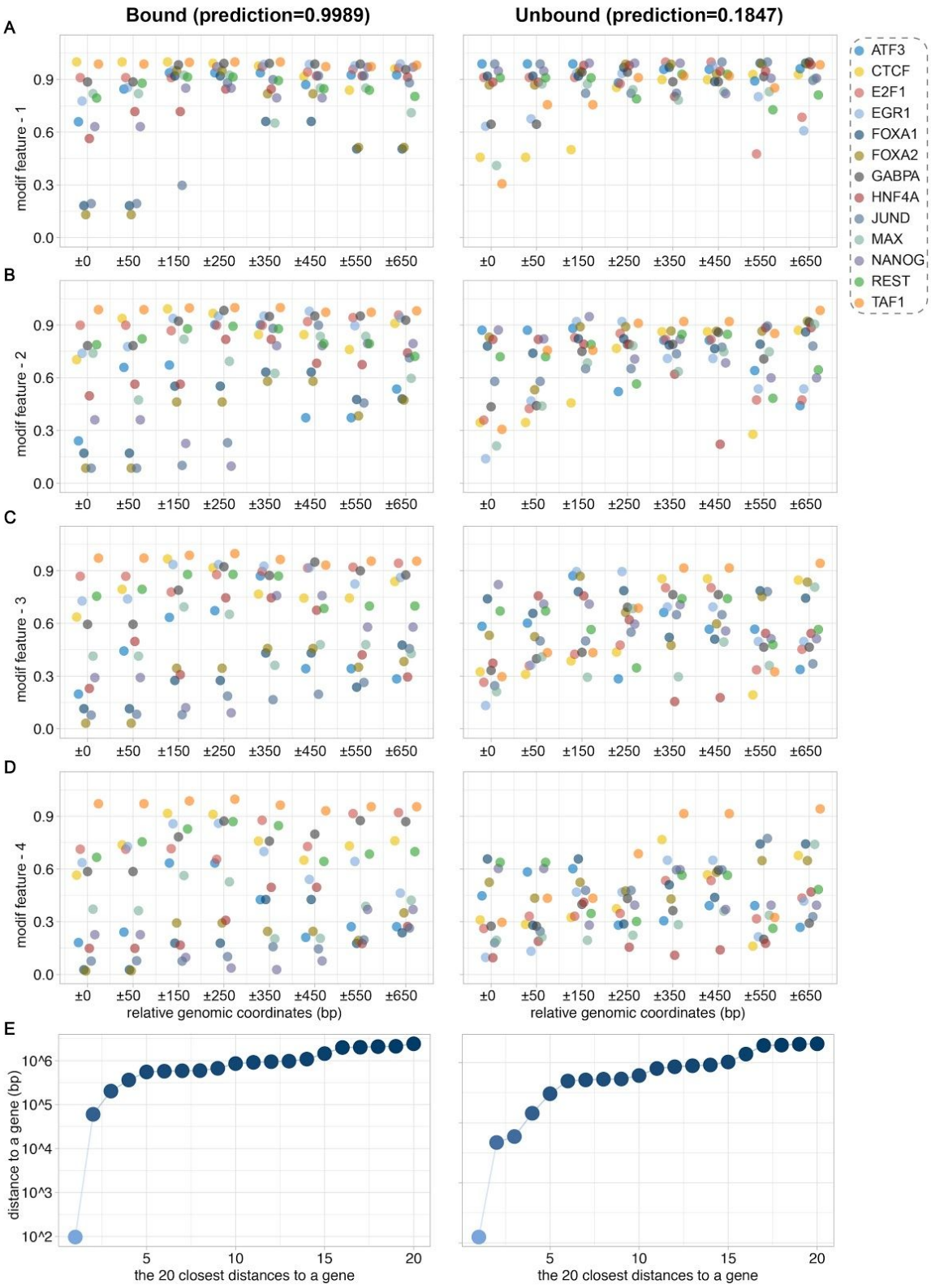


Supplemental Fig. S13. Genome-wide quantile normalization of DNase-seq signals across cell types.

The original genome-wide DNase-seq read depth of multiple technical and biological replicates from the same cell type are summed and ranked in the decreasing order. Each unique read depth of the training cell type is quantile normalized to the corresponding (average) value of the testing cell type, to reduce the cell type-specific cleavage and sequencing biases. After quantile normalization, the signals are natural-log-transformed and re-ordered back to the original profile.

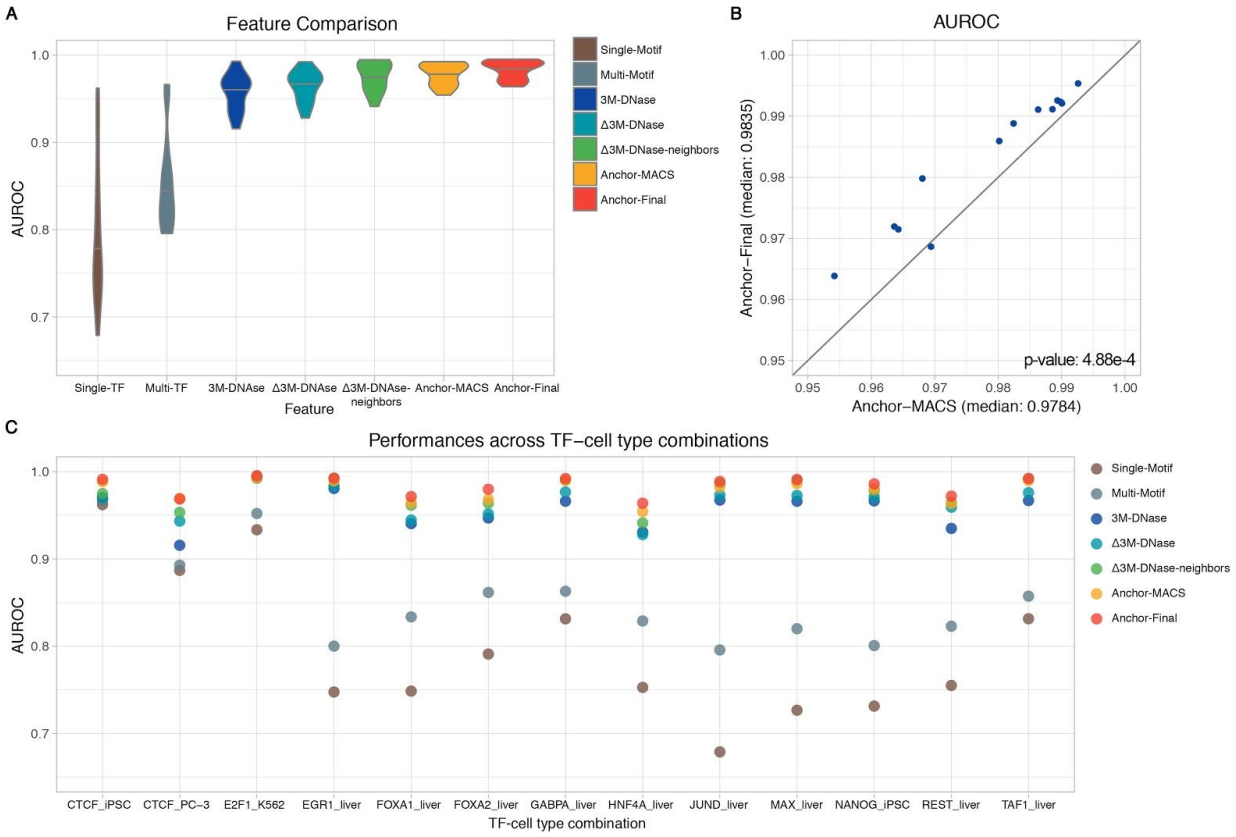


Supplemental Fig. S14. The read alignment coverage signals and DNase-seq features of two example "Bound" and "Unbound" genomic intervals for CTCF in the IMR-90 cell line. The "Bound" interval (left) locates between 120,100 and 120,300 in Chr 10, and the "Unbound" interval (right) locates between 179,150 and 179,350 in Chr 10. **A**. The original read alignment coverage signals in the 200bp interval at the base pair-level resolution. **B**. The 45 bin-level "3M-DNase-neighbors" features (maximum, mean and minimum) around the center of interest. **C**. The 45 bin-level "Δ3M-DNase-neighbors" features (Δmaximum, Δmean and Δminimum) around the center of interest. **D**. The 30 bin-level frequency and Δfrequency features.



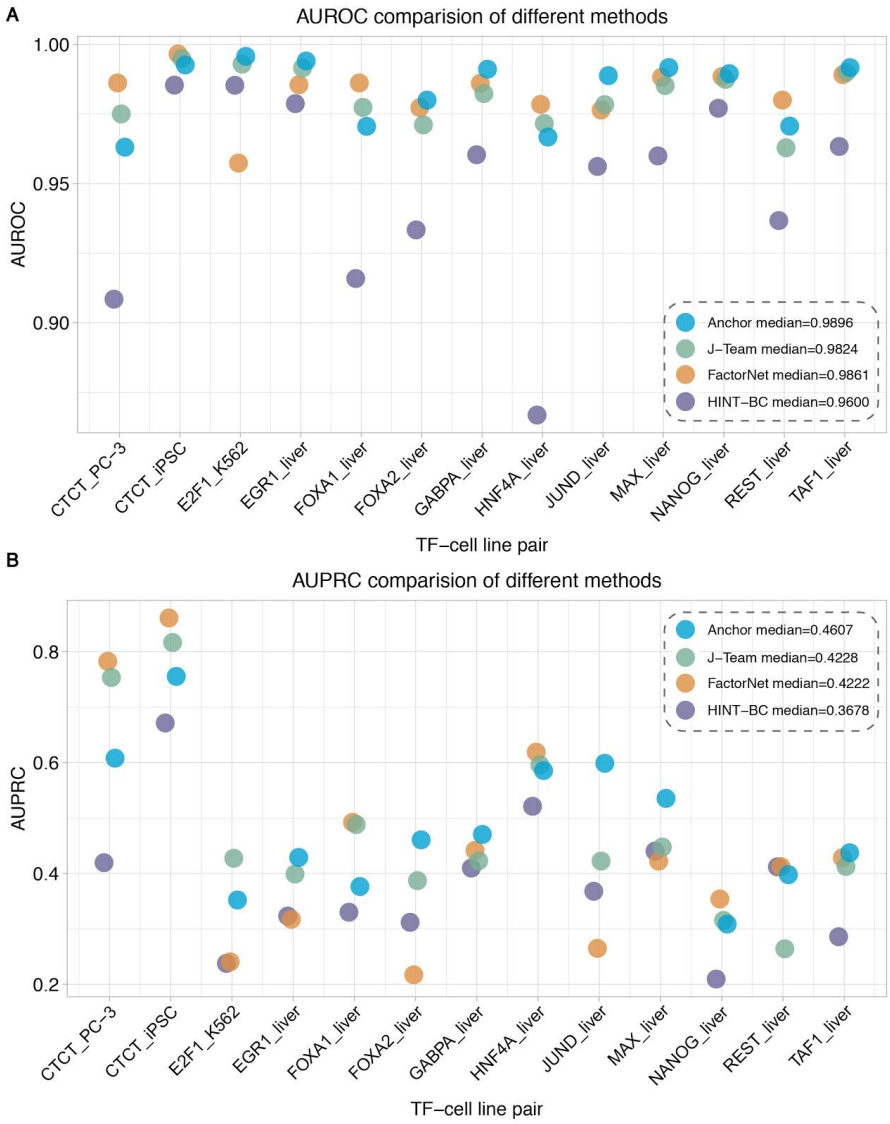
Supplemental Fig. S15. The motif-based and distance-to-gene features of two example "Bound" and "Unbound" genomic intervals for CTCF in the IMR-90 cell line.

The “Bound” interval (left) locates between 120,100 and 120,300 in Chr 10, and the “Unbound” interval (right) locates between 179,150 and 179,350 in Chr 10. For each genomic position, the 13 TF matching scores were calculated via matrix multiplication between PWMs and human genome sequence chunks in either strand orientation. The top 4 scores with each 200bp genomic interval and neighbors were selected as features shown in **A-D**, respectively. The top 20 closest distances to a gene were calculated based on the Gencode annotations as 20 distance-to-gene features in **E**.



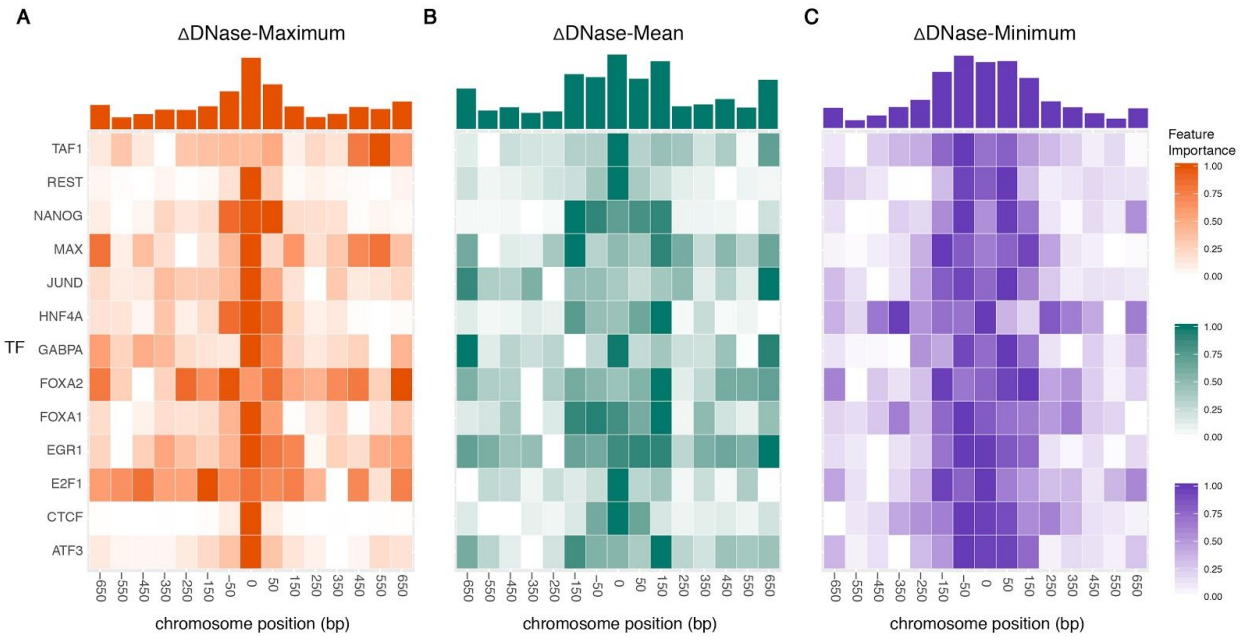
Supplemental Fig. S16. The comparison of different features in the 13 testing TF-cell type pairs.

The predictive performances of two sequence-based models (Single-TF and Multi-TF), three DNase-based models (3M-DNase, Δ3M-DNase and Δ3M-DNase-neighbors) and two ensemble models (Anchor-MACS and Anchor-Final). **A.** The median AUROCs of models using different types of features. **B.** The AUROC comparison of the Anchor-MACS and Anchor-Final models in the 13 TF-cell type pairs shown as blue dots. If a dot is above the diagonal line, it means the AUROC of the Anchor-Final model is higher. The paired Wilcoxon signed-rank test was performed between the predictions of these two models and the p-value equals 4.88e-4. **C.** The complete AUROCs of the 13 testing TF-cell type pairs.

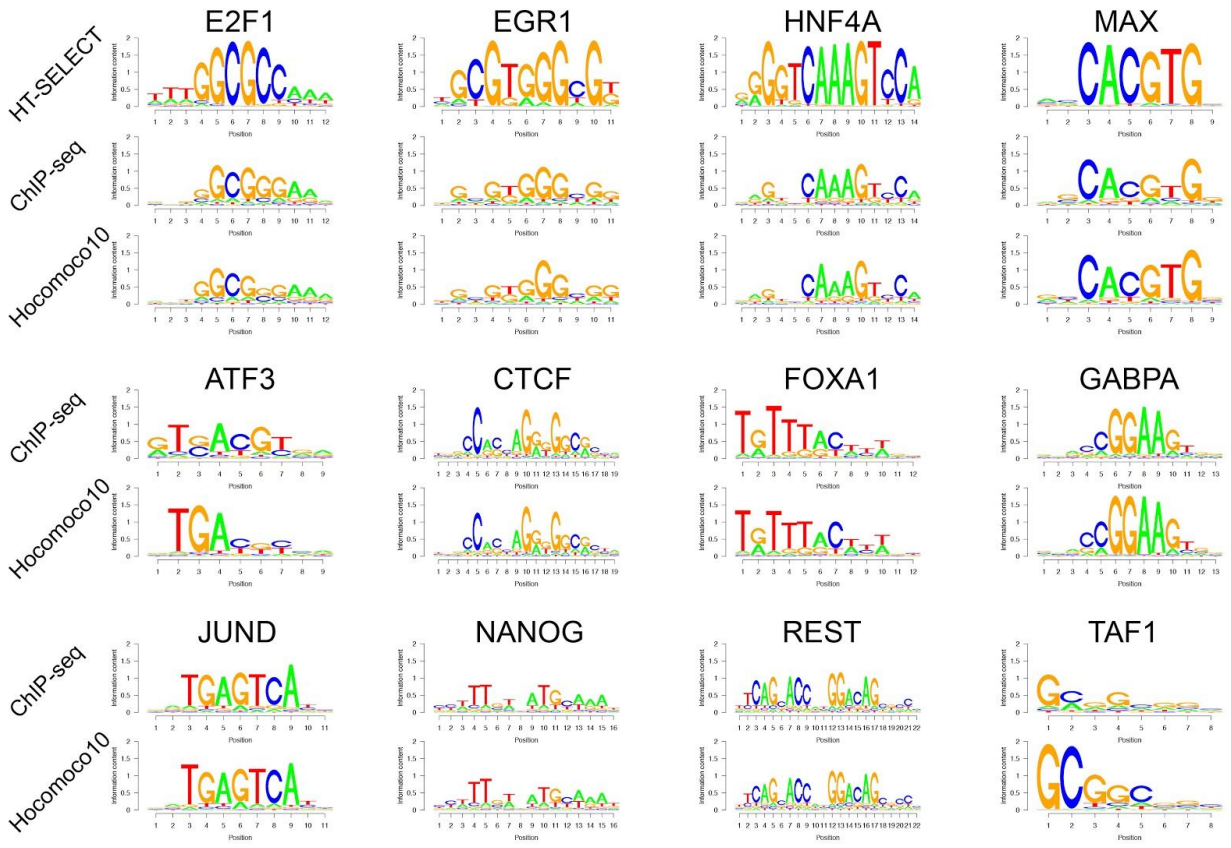


Supplemental Fig. S17. The comparison of different methods in the 13 testing TF-cell type pairs.

The prediction **A**. AUROCs and **B**. AUPRCs of four methods are shown in different colors. The median values across the 13 TF-cell pairs are highlighted. The four methods are: 1. Anchor 2. the method developed by J-team (Keilwagen et al. 2017) 3. FactorNet (Quang and Xie 2017) 4. HINT-BC (Gusmao et al. 2016).



Supplemental Fig. S18. The spatial distributions of Δ3M-DNase feature importances
The importance heatmaps of **A.** "Δmaximum" **B.** "Δmean" and **C.** "Δminimum" features display common and TF-specific distributions in the vicinity of the TF binding site. The corresponding accumulated feature importances are shown as histograms above the heatmaps.



Supplemental Fig. S19. TF motif comparison using different sources.

We have compared TF motifs generated from different assays and methods. The “Hocomoco 10” (http://hocomoco10.autosome.ru/) motifs were used in our Anchor framework. The “HT-SELEX” motifs were verified in vitro by HT-SELEX. They were downloaded from JASPAR website and only four TFs are available (http://jaspar.genereg.net/). The “ChIP-seq” motifs were verified in vivo by ChIP-seq. They were downloaded from the new version Hocomoco 11, which generated motifs based on ChIP-seq experiments instead of computation models (http://hocomoco11.autosome.ru/).

Supplemental Table Legends

Supplemental Table S1. The partition of the training chromosomes into two sets and the held-out set

Supplemental Table S2. The training and testing cell types of the 13 TFs

Supplemental Table S3. The AUROCs and AUPRCs of the 43 training within-cell and the 13 testing cross-cell TF-cell pairs.

Supplemental Table S4. Multiple evaluate metrics at 12 different cutoffs.

Supplemental Table S5. The AUROCs and AUPRCs of the top 4 teams in the ENCODE-DREAM challenge.
For each TF-cell pair, the cross-cell performance, within-cell performance and the difference are calculated. The team that has the smallest performance difference is shown in bold. The median values of all 13 TF-cell pairs are shown in green. These AUROCs and AUPRCs are from:
https://www.synapse.org/#!Synapse:syn6131484/wiki/405275
https://www.synapse.org/#!Synapse:syn6131484/wiki/412905

Supplemental Table S6. The prediction performance and cell line tiers of the 55 ChIP-seq Data downloaded from ENCODE project.
All the ChIP-seq data were downloaded from ENCODE project website:
https://www.encodeproject.org/.
The Accession numbers are provided in the last column. The DNase-seq data were downloaded from the ftp portal:
http://hgdownload.soe.ucsc.edu/goldenPath/hg19/encodeDCC/wgEncodeUwDnase/

Supplemental Table S7 The prediction performances using DNase-seq data within and outside ENCODE.
The DNase-seq data from the ENCODE project were downloaded from:
https://www.synapse.org/#!Synapse:syn6112317
The DNase-seq data from the Roadmap project were downloaded from:
https://www.encodeproject.org/experiments/ENCSR794OFW/
https://www.encodeproject.org/experiments/ENCSR477RTP/

Supplemental Table S8. The AUROCs and AUPRCs of models using different types of features.
For each TF-cell pair, the model achieves the highest score is shown in bold. The median values of all 13 TF-cell pairs are shown in green.

Supplemental Table S9. The importance of the 556 features in predicting the 13 TF-cell line pairs

Supplemental Table S10. The TF superclass and class of the 13 TFs

TFs belong to the same superclass are shown in the same color. The classification was obtained from http://www.edgar-wingender.de/huTF_classification.html