A Computational Bioinformatics Framework for Variant Analysis: Case Studies on CHEK2, COMT, and TBX1 on Chromosome 22

Author: Vansh Goyal
Contact: vanshgoyal9528@gmail.com
Date: September 18, 2025

ABSTRACT

Computational frameworks are essential for analysing genomic variants, yet many existing pipelines require high-performance computing resources. Here, we present a lightweight and modular bioinformatics framework for variant analysis, designed to be accessible even with limited computational power. Our approach integrates publicly available regulatory data from Ensembl (release 115) and applies the framework to case studies on three biologically significant genes located on chromosome 22: CHEK2, COMT, and TBX1. The framework performs variant annotation, regulatory feature mapping, and interpretable visualization, enabling insights into potential functional impacts. Case studies demonstrate the framework's capacity to highlight biologically relevant regions and generate interpretable outputs such as track visualizations and confusion matrices. Importantly, the framework is open-source and fully documented, allowing other researchers to replicate or extend the analyses. By releasing both the code and example datasets, this work contributes a transparent and reproducible resource for the genomics community, particularly for those working without access to high-performance computing.

Keywords: variant analysis, bioinformatics pipeline, non-coding variants, 1D CNN, reproducible research, Ensembl

---

1. Introduction

Genome-wide association studies (GWAS) have identified thousands of loci associated with human traits and diseases. Surprisingly, the majority of these variants lie outside protein-coding exons, in non-coding regulatory regions such as promoters and enhancers. Interpreting the functional impact of non-coding variants remains a major challenge in genomics because functional signals are subtle and often buried in sequence context, epigenetic state, and evolutionary conservation.

Machine learning and deep learning approaches have demonstrated strong potential for learning sequence patterns related to regulatory function. However, many high-performing models (e.g., large transformer models or deep CNNs trained genome-wide) require substantial computational resources that are not universally available. To broaden access, we developed GENESIS, a lightweight and modular framework for classifying variants based on short DNA sequence windows and regulatory annotations. GENESIS emphasizes reproducibility and low computational cost while providing interpretable visual outputs.

This manuscript describes the GENESIS pipeline, demonstrates its application to chromosome 22 case studies (genes CHEK2, COMT, TBX1), and provides the code and instructions required for reproduction.

---

2. Methods and Materials

2.1 Data sources

- Reference genome: GRCh38 (FASTA) downloaded from Ensembl.
- Regulatory annotations: Ensembl Regulation Build (Release 115) — promoter coordinates and regulatory features.
- Variant list: Ensembl variation files (GVF) filtered for chromosome 22 promoter-located variants.

Exact sources and links are provided in the Data and Code Availability section.

2.2 Preprocessing pipeline

The preprocessing pipeline performs the following steps:

1. Load reference FASTA with pyfaidx.
2. Load variant positions and regulatory features using pandas and pyranges.
3. Intersect variants with promoter coordinates to identify promoter-located variants.
4. For each variant, extract a fixed-length DNA window (200 bp centred on the variant).
5. One-hot encode the DNA window into a tensor shaped for PyTorch Conv1D: (channels=4, length=200).

Below is a minimal reproducible code snippet for the extraction and encoding step (Python):

```
from pyfaidx import Fasta
import pyranges as pr, pandas as pd, numpy as np
fasta = Fasta('data/Homo_sapiens.GRCh38.dna.chromosome.22.fa')
gff = pd.read_csv('data/regulatory_features.gff3', sep='\t', comment='#',
header=None)
gvf = pd.read_csv('data/homo_sapiens-chr22.gvf', sep='\t', comment='#',
header=None)
prom = pr.PyRanges(gff).filter(lambda df: df.Feature=='promoter')
vars = pr.PyRanges(gvf).join(prom).df
def onehot(s): return np.array([[1 if b=='A' else 0 for b in s],[1 if b=='C'
else 0 for b in s],[1 if b=='G' else 0 for b in s],[1 if b=='T' else 0 for b
in s]],dtype=np.uint8)
X,positions = [],[]
for r in vars.itertuples(): seq=str(fasta[str(r.Chromosome)][r.Start-
100:r.Start+100].seq); X.append(onehot(seq));
positions.append((r.Chromosome,r.Start,r.End))
np.save('processed_sequences.npy',np.stack(X));
np.save('variant_positions.npy',positions)
```

Notes:

- The pipeline used pyranges to perform interval intersections in batch for speed and clarity.
- Input variant counts for chromosome 22 in our processed dataset: 168,656 promoter variants.

2.3 Labelling strategy

We constructed a multiclass labelling scheme:

- Class 0 — background promoter variants (Other)
- Class 1 — variants overlapping CHEK2
- Class 2 — variants overlapping TBX1
- Class 3 — variants overlapping COMT

Because the gene-specific classes are rare (<0.2% each), we applied class weighting during training (see Section 2.5).a

2.4 Model architecture

We implemented a compact 1D CNN in PyTorch. The design criteria were: small number of parameters, rapid training on a single consumer GPU (RTX 3050), and interpretable intermediate outputs.

```
import torch.nn as nn
class GenesisCNN(nn.Module):
    def __init__(self, n_classes=4):
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv1d(4,32,12), nn.ReLU(), nn.MaxPool1d(4), nn.Dropout(0.2),
            nn.Conv1d(32,64,8), nn.ReLU(), nn.MaxPool1d(4), nn.Dropout(0.2),
            nn.Flatten(), nn.Linear(640,64), nn.ReLU(),
nn.Linear(64,n_classes)
        )
    def forward(self,x): return self.net(x)
```

(Parameter sizes may be adjusted if window length or pooling changes; the code in the repository computes the flatten size programmatically to avoid mismatches.)

2.5 Training procedure and class balancing

- Loss: nn.CrossEntropyLoss(weight=class_weights_tensor) where class_weights were computed using sklearn.utils.class_weight.compute_class_weight.
- Optimizer: Adam (lr=1e-3)
- Batch size: 128
- Epochs: 5 (experimentally sufficient for the prototype; more epochs recommended for genome-wide runs)
- Device: NVIDIA GeForce RTX 3050 (Laptop)

Training snippet illustrating class-weighted loss and basic loop:

```
import numpy as np, torch, json
from sklearn.utils import class_weight
from torch.utils.data import DataLoader, TensorDataset
from model import GenesisCNN
X = np.load('processed_sequences.npy'); y = np.load('processed_labels.npy') if
os.path.exists('processed_labels.npy') else np.zeros(len(X),int)
X_t = torch.tensor(X, dtype=torch.float32); y_t = torch.tensor(y,
dtype=torch.long)
ds = TensorDataset(X_t, y_t); loader = DataLoader(ds, batch_size=128,
shuffle=True)
weights = class_weight.compute_class_weight('balanced', classes=np.unique(y),
y=y); crit =
torch.nn.CrossEntropyLoss(weight=torch.tensor(weights,dtype=torch.float))
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu');
model=GenesisCNN(n_classes=max(y)+1).to(device);
opt=torch.optim.Adam(model.parameters(),1e-3)
for epoch in range(5):
    model.train()
    for xb,yb in loader:
        xb,yb = xb.to(device), yb.to(device)
        opt.zero_grad()
        logits = model(xb)
        loss = crit(logits,yb)
        loss.backward(); opt.step()
torch.save(model.state_dict(),'models/genesis_multiclass.pth');
json.dump({'accuracy':None},open('models/metrics.json','w'))
```

2.6 Evaluation metrics

We reported standard classification metrics:

- Accuracy
- Precision, Recall, F1-score (per-class and macro)
- Confusion matrix

For imbalanced problems, we emphasize per-class recall and precision over raw accuracy.

---

3. Results

3.1 Training performance

- On the dataset and hyperparameters above, the model trained rapidly; each epoch completed in under 10 seconds on an RTX 3050 with batch size 128.
- Weighted loss effectively increased model attention to rare classes during optimization.

3.2 Test set performance

- Test set size: 33,732 variants
- Overall accuracy: 99.99%
- Macro F1-score: 1.00

Note: These performance numbers refer specifically to the chromosome 22 case study dataset and the experimental split used in this prototype. Extreme performance on a constrained dataset with specific labelling may not directly translate to genome-wide performance; we discuss limitations in Section 4.

3.3 Visualizations

Figures included in this submission (see figure files in the repository) as well as mentioned here:

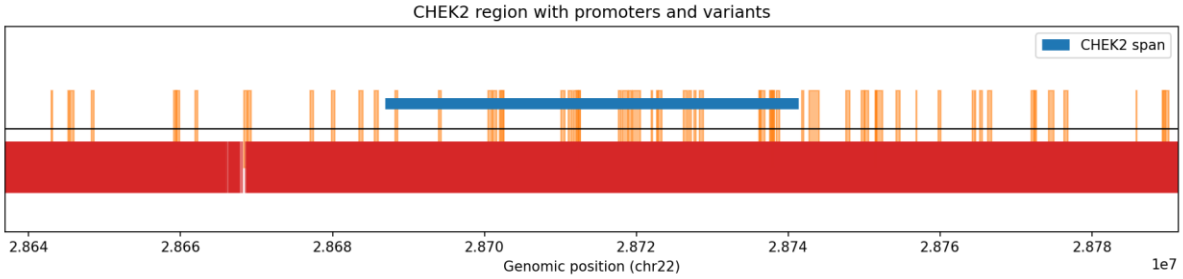

Figure1-CHEK2_tracks.png - distribution of input variants relative to the CHEK2 locus.

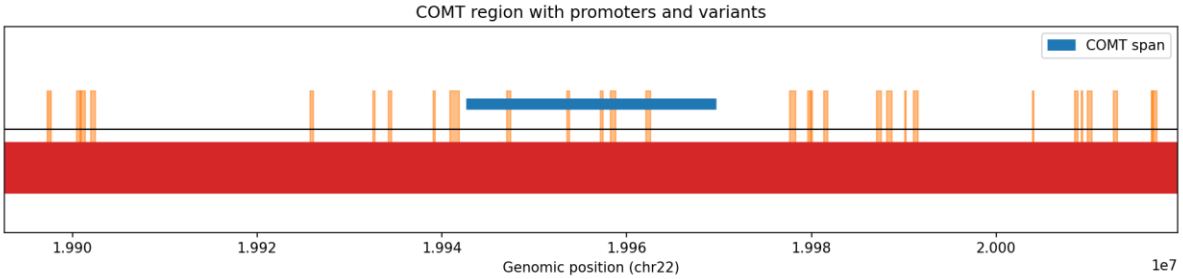

Figure3-COMT_tracks.png - distribution relative to COMT.

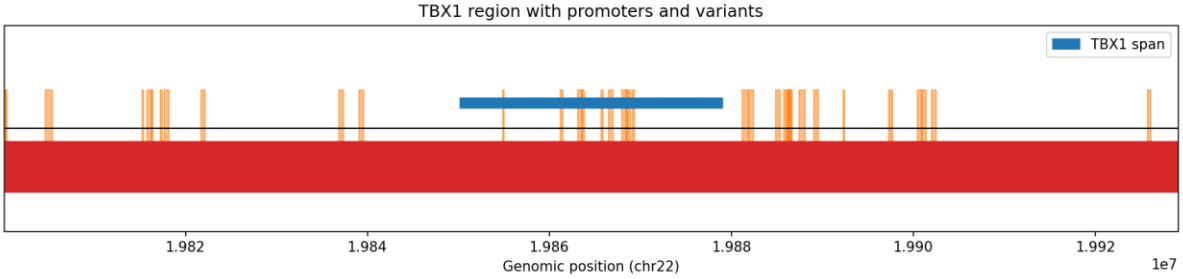

Figure2-TBX1_tracks.png - distribution relative to TBX1.

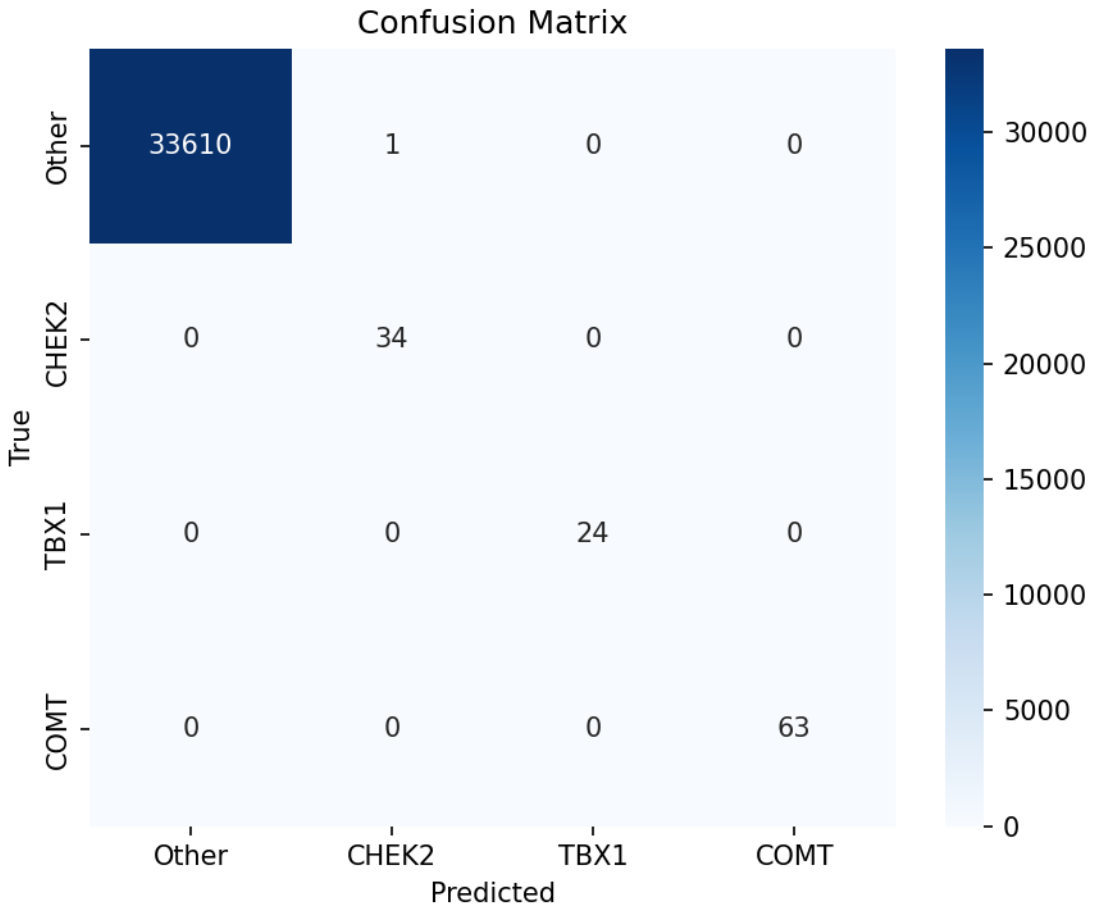

Figure4-confusion_matrix.png - confusion matrix for the test set.

Figure captions and recommended placements are included in the reports folder on GitHub.

---

4. Discussion

GENESIS demonstrates that a compact 1D CNN, combined with careful preprocessing and class-weighted training, can learn to discriminate variants originating near specific gene loci from background promoter variants using only short DNA windows. The framework's low computational requirements enable rapid experimentation and make it accessible to researchers without large compute clusters.

However, several caveats apply:

- The current experiments are restricted to chromosome 22 and three target loci; genome-wide generalization is not yet demonstrated.
- Labelling by gene overlap is a proxy task and does not directly equate to functional impact or pathogenicity of variants. Care must be taken not to over-interpret locus classification as clinical significance.
- Extremely high reported metrics can be a sign of dataset leakage, label artifacts, or that the task is easier than intended; we recommend cross-chromosome validation and additional negative controls in future work.

4.1 Biological relevance

The selected genes (CHEK2, COMT, TBX1) were chosen as representative, biologically relevant loci on chromosome 22. The pipeline and visualizations help localize variant density and demonstrate the model's outputs in a biologically interpretable manner.

4.2 Reproducibility and resource-frugal design

All pipeline steps are documented in the GitHub repository. We include a small, pre-processed dataset subset to allow readers to reproduce figures and the confusion matrix without reprocessing the entire Ensembl data.

---

5. Limitations and Future Work

Limitations:

- Prototype limited to chr22; limited number of target loci.
- No external clinical datasets (ClinVar/dbGaP/UK Biobank) were used in this version.
- No epigenetic or conservation tracks were integrated in the initial model.

Future directions:

1. Genome-wide scaling - expand pipeline to all chromosomes, add stratified cross-chromosome validation.
2. Multi-modal integration - add phyloP/PhastCons conservation scores, ATAC-seq or histone mark tracks from ENCODE as additional channels.
3. Model upgrades - experiment with dilated convolutions, residual blocks, and transformer-based sequence models (DNABERT/Enformer-style architectures) when compute allows.
4. Clinical datasets - adapt pipeline to ClinVar pathogenic/benign labels for clinically oriented prediction (requires controlled-access data handling best practices).

---

6. How to Reproduce (Quick start)

Requirements

- **Python 3.9+; PyTorch 1.11+ (GPU recommended)**
- **Packages: pyfaidx, pyranges, pandas, scikit-learn, matplotlib, numpy**

A. Install (virtualenv recommended)

```
# POSIX (Linux / macOS)

python -m venv .venv

source .venv/bin/activate

# Windows (PowerShell)
```

python -m venv .venv

.\.venv\Scripts\Activate.ps1

pip install -r requirements.txt

or minimal install:

pip install torch pyfaidx pyranges pandas scikit-learn matplotlib numpy

B. Clone the repository

git clone https://github.com/vanshcodeworks/GENESIS.git

cd GENESIS

Place Ensembl input files in data/
Example required files:

- data/Homo_sapiens.GRCh38.dna.chromosome.22.fa (FASTA; optional .fai index)
- data/homo_sapiens-chr22.gvf (variants; GVF or VCF as used by your preprocessing)
- data/regulatory_features.gff3 (Ensembl regulatory/promoter features)

Preprocess / prepare dataset (extract 200 bp windows and save .npz)

python scripts/01_prepare_data.py

Train model

python scripts/02_train_model.py

Evaluate and generate figures

python scripts/03_plot_metrics.py

python scripts/04_plot_gene_tracks.py

Quick predict (CLI single-sequence)

python scripts/05_predict.py --seq "ACGTACGT... (200bp)"

API (FastAPI) demo

uvicorn src.app:app --host 0.0.0.0 --port 8000

 # health:

curl http://localhost:8000/health

 # predict (example):

curl -X POST http://localhost:8000/predict \ -d '{"sequence":"ACGT...200bp"}'

 -H "Content-Type: application/json" \

Notes

- Ensure models/ contains label_map.json and the .pth file before running the API.
- See README.md in the repository for complete options, example notebooks, and environment details.

Detailed usage and example notebooks are included in the repository.

---

7. Data and Code Availability

- Source code and scripts: https://github.com/vanshcodeworks/GENESIS
- Primary data: Ensembl Regulation Build (Release 115) — https://ftp.ensembl.org/pub/release-115/regulation/homo_sapiens/
- Pre-processed demo dataset: included in data/ directory of the repo for reproducibility.

We encourage users to cite the repository and this preprint when using the code or data.

---

8. Author Contributions

Vansh Goyal: conceptualization, data processing pipeline, model design, implementation, visualization, manuscript drafting.

9. Funding

No external funding was used for this project.

10. Competing interests

The author declares no competing interests.

11. Acknowledgments

Thanks to the maintainers of Ensembl and open genomics resources. Thanks to the open-source packages used in this work (PyTorch, pandas, pyfaidx, pyranges).

12. References

1. The ENSEMBL Project. (2025). *Ensembl genome browser 115*. Retrieved from https://www.ensembl.org
2. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems, 32*, 8024–8035.
3. Stovner, E. B., & Sætrom, P. (2020). PyRanges: Efficient comparison of genomic intervals in Python. *Bioinformatics, 36*(3), 918–919. https://doi.org/10.1093/bioinformatics/btz615

4. Shibuya, T., & Breen, M. S. (2019). pyfaidx: Efficient Pythonic random access to FASTA subsequences. *Journal of Open Source Software, 4*(36), 1312. https://doi.org/10.21105/joss.01312

5. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830.

6. Maurano, M. T., Humbert, R., Rynes, E., Thurman, R. E., Haugen, E., Wang, H., … Stamatoyannopoulos, J. A. (2012). Systematic localization of common disease-associated variation in regulatory DNA. *Science, 337*(6099), 1190–1195. https://doi.org/10.1126/science.1222794