**Supplementary Information**

**ARCLID: Accurate and Robust Characterization of Long Insertions and Deletions in Genome**

# Deep Learning Models

The deep learning model used in this study to detect structural variants (SVs) is YOLOv11x, which contains approximately 56 million parameters. We used Ultralytics [1] and pytorch libraries in python to train the deep learning model. YOLOv11x is organized into three main components: the backbone, the neck, and the head. During the forward pass, an input image is processed through a series of convolutional layers and CSP-based backbone blocks [2]. These initial layers extract hierarchical features by applying learned filters that capture edges, textures, and higher-order patterns. The backbone employs cross-stage partial (CSP) connections to reduce computational redundancy while preserving feature richness. In addition, YOLOv11x incorporates attention mechanisms, which improve feature selection and representation compared to earlier versions such as YOLOv8, thereby enhancing the model’s ability to focus on relevant regions. As the image traverses the backbone, spatial resolution decreases while channel depth increases, enabling the network to capture more abstract semantic information. Intermediate feature maps are then passed to the neck (e.g., PANet), which fuses multi-scale features through upsampling and concatenation, strengthening the model’s capability to localize objects of varying sizes.

In the detection head, YOLO generates predictions by applying anchor-free detection mechanisms. Instead of predefined anchor boxes, the model directly predicts bounding box coordinates (x, y, width, height), objectness scores, and class probabilities at three distinct scales (e.g., 80x80, 40x40, 20x20 grids). Each grid cell’s output tensor combines localization (via CIoU-aware regression), confidence (probability of containing an object), and classification scores. The model employs a combination of convolutional layers and activation functions (e.g., SiLU) to refine these predictions. Finally, the outputs from all scales are aggregated, and non-maximum suppression (NMS) is applied post-forward pass to filter overlapping detections, yielding the final set of bounding boxes, classes, and confidence scores. This streamlined process ensures real-time efficiency while maintaining high detection accuracy across diverse object sizes.

YOLO employs three primary cost functions to optimize its object detection performance [3]: box loss (CIoU for bounding box regression, weight=7.5), classification loss (BCE for class prediction, weight=0.5), and distribution focal loss (DFL) for precise bounding box distribution modeling (weight=1.5). These default weights, defined in the hyperparameter configuration, balance the contributions of each component during training, ensuring accurate localization, classification, and confidence estimation. The values are tuned to prioritize bounding box accuracy (highest weight for CIoU) while maintaining a balance between detecting objects (objectness) and refining class predictions.

**Box Regression Loss (CIoU).** The Complete Intersection over Union (CIoU) loss enhances the traditional IoU metric by introducing penalties for both the distance between box centers and discrepancies in aspect ratios [3]. Its formula is shown in below.

is the Euclidean distance between box centers, is the diagonal length of the enclosing box C, is the penalty for aspect ratio mismatch, and *α* balances the and center distance terms.

**DFL.** Designed to refine bounding box localization [3], DFL treats coordinates as discrete distributions and emphasizes probabilities near the target value:

Where  is the continuous target value (e.g., box coordinate), and *​,*  are the nearest discrete bins with probabilities *, and* ​.

**Classification Loss.** This loss function [3] handles multi-class prediction by computing the binary cross-entropy between ground truth labels and predicted class probabilities:

Where is the number of classes,  is the binary groundtruth label (0 or 1) for class , and ​ is the model’s predicted probability for that class.

**Total Loss.** The combined loss is a weighted sum of the individual components as below:

**Training process.** As mentioned in the paper, two types of pileup versions are generated, one type with the size of 5000 bp to detect SVs smaller than 2Kbp, and another one with the size of 50000 bp to detect SVs larger than 2Kbp. Therefore, two YOLOv11x were trained for each type of pileup image with 200 epochs and batch size of 16. The training process curves are presented in Figure 1 and 2.

A group of graphs showing the results of a graph

AI-generated content may be incorrect.

Figure 1. Training curves. a) Training curves for SVs smaller than 2 Kbp. b) Training curves for SVs larger than 2 Kbp. A closer examination of the curves shows that the model struggles more with detecting SVs >2 Kbp compared to the model trained on smaller SVs (<2 Kbp). The model trained on larger SVs achieves only about 0.70 mAP@50–95, whereas the model trained on smaller SVs reaches approximately 0.90 mAP@50–95. It is important to note that mAP (mean Average Precision) is a widely used evaluation metric in computer vision for object detection tasks.

A diagram of different colored lines

AI-generated content may be incorrect.

Figure 2. Precision–recall curves, F1 curves, and confusion matrices of the two trained models on validation set. a) Metrics for the model trained on smaller SVs (<2 Kbp). b) Metrics for the model trained on larger SVs (>2 Kbp). We observe that very large SVs (b) pose greater challenges for the deep learning model: at its best performance, the model detects SVs >2 Kbp with an F1-score of 0.72 at a confidence threshold of 0.417. In contrast, smaller SVs (<2 Kbp) are easier to detect, achieving an F1-score of 0.93 at a lower confidence threshold of 0.284 (indicating higher F1 at lower confidence). Closer inspection of the precision–recall curves reveals that insertion events are generally more difficult to detect than deletions. Moreover, within insertions, heterozygous insertions present a greater challenge compared to homozygous insertions.

# Resource Usage Evaluation

Table 1. Resource usage evaluation on HG002 28X. To conduct the evaluation, we fixed the number of CPU cores and threads. SVIM does not support multi-threading and operates exclusively with a single core and thread. Also, Sniffles2 failed to complete the process when configured to use only one core and one thread.

.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Core/Thread** | **Tool** | **Wall-Clock (s)** | **CPU Time (s)** | **Memory (Mb)** |
| 1/1 | ARCLID | 12128 | 10836 | 126 |
| Sniffles2 | × | × | × |
| CuteSV | 3536 | 2987 | 107 |
| SVIM | 5342 | 4012 | 670 |
| pbsv | 7203 | 7158 | 7314 |
| 4/4 | ARCLID | 1928 | 5324 | 585 |
| Sniffles2 | 501 | 1319 | 692 |
| CuteSV | 352 | 1167 | 171 |
| SVIM | × | × | × |
| pbsv | 4244 | 6836 | 10198 |
| 8/8 | ARCLID | 1032 | 6348 | 962 |
| Sniffles2 | 217 | 1127 | 853 |
| CuteSV | 269 | 1226 | 270 |
| SVIM | × | × | × |
| pbsv | 3664 | 6518 | 10794 |
| 16/16 | ARCLID | 824 | 8352 | 1437 |
| Sniffles2 | 164 | 1153 | 1307 |
| CuteSV | 212 | 1750 | 524 |
| SVIM | × | × | × |
| pbsv | 3551 | 7168 | 7574 |

# Execution Parameters

**Truvari relaxed mode:**

truvari bench -b truth\_set.vcf.gz -c pred.vcf.gz -o truvari\_results --includebed high\_confidence\_regions.bed --no-ref b --refdist 1000 --pctseq 0 --passonly

**Truvari strict mode:**

truvari bench -b truth\_set.vcf.gz -c pred.vcf.gz -o truvari\_results --includebed high\_confidence\_regions.bed --no-ref b --refdist 50 --pctsize 0.9 --pctovl 0.9 --pctseq 0 --passonly

**Pbmm2:**

pbmm2 align $ref\_idx $fq\_all $alignment --num-threads 16 --sort --sample $sample\_name --preset CCS

**Sniffles2:**

sniffles --input aligned\_reads.bam --vcf output.vcf --genotype-ploidy 2

**cuteSV:**

We set cuteSV parameters along with different min support read given the coverage of the sample using the original paper and GitHub of cuteSV. min\_support for different coveages: 28X → 6, 20X → 4, 10X → 3, 5X → 2

cuteSV aligned\_reads.bam output.vcf ./ --sample sample\_name --max\_cluster\_bias\_INS 1000 --diff\_ratio\_merging\_INS 0.9 --max\_cluster\_bias\_DEL 1000 --diff\_ratio\_merging\_DEL 0.5 --min\_support min\_sup

**SVIM:**

svim alignment ./ aligned\_reads.bam ref.fa --sample HG002 --type DEL,INS --cluster\_max\_distance 1.4

cat ./variants.vcf | grep -v 'SUPPORT=1;\|SUPPORT=2;\|SUPPORT=3;\|SUPPORT=4;\|SUPPORT=5;\|SUPPORT=6;' > ./variants\_filtered.vcf

**PBSV:**

pbsv discover aligned\_reads.bam ./vars.svsig.gz --tandem-repeats tnd\_file.bed

pbsv call ref.fa ./vars.svsig.gz variants.vcf -t INS,DEL

**VISOR:**

The entire process of using VISOR to generate simulated reads

1. Rscript /home/user/visor\_app/VISOR/scripts/randomregion.r -h
2. cut -f1,2 small.reference.fa.fai > chrom.dim.tsv
3. Rscript /home/user/visor\_app/VISOR/scripts/randomregion.r -d chrom.dim.tsv -n 15000 -l 350 -s 50 -x exclude.bed -v 'insertion,deletion,inversion,inverted tandem duplication,translocation copy-paste,reciprocal translocation' -r '40:40:5:5:5:5' | sortBed > HACk.random.bed
4. VISOR HACk -g small.reference.fa -b HACk.random.bed -o hack.3.out
5. VISOR LASeR -g small.reference.fa -s hack.3.out -b shorts.laser.simple.bed -o laser.1.out --threads 4 --coverage 30 --length\_mean 13000 --length\_stdev 5000 --tag --fastq --compress --read\_type pacbio --error\_model pacbio2016 --qscore\_model pacbio2016

# References

[1] Glenn Jocher and Jing Qiu, “Ultralytics, YOLOv11”, 2024. <https://github.com/ultralytics/ultralytics>

[2] Rahima Khanam and Muhammad Hussain, “YOLOv11: An Overview of the Key Architectural Enhancements,” *ArXiv*, 2024. <https://arxiv.org/abs/2410.17725>

[3] Soumyadip, “YOLO loss function, Part1 and Part2,” 2024. <https://learnopencv.com/yolo-loss-function-siou-focal-loss/>