

# Supplementary Material for

## Accurate, scalable structural variant genotyping in complex genomes at population scales

Ming Hu<sup>1,6</sup>, Penglong Wan<sup>1,6</sup>, Chengjie Chen<sup>2,6</sup>, Shuyuan Tang<sup>1</sup>, Jiahao Chen<sup>1</sup>, Liang Wang<sup>1</sup>, Mahul Chakraborty<sup>3</sup>, Yongfeng Zhou<sup>2</sup>, Jinfeng Chen<sup>4</sup>, Brandon S. Gaut<sup>5</sup>, J.J. Emerson<sup>5,†</sup> & Yi Liao<sup>1,†</sup>

<sup>1</sup> Key Laboratory of Biology and Genetic Improvement of Horticultural Crops (South China), Ministry of Agriculture and Rural Affairs, College of Horticulture, South China Agricultural University, Guangdong 510642, China

<sup>2</sup> Tropical Crops Genetic Resources Institute, Chinese Academy of Tropical Agricultural Sciences & National Key Laboratory for Tropical Crop Breeding & Laboratory of Crop Gene Resources and Germplasm Enhancement in South China, Ministry of Agriculture and Rural Affairs & Key Laboratory of Tropical Crops Germplasm Resources Genetic Improvement and Innovation of Hainan Province, Hainan, 571101, China

<sup>3</sup> Department of Biology, Texas A&M University, College Station, TX, 77843, USA

<sup>4</sup> State Key Laboratory of Integrated Management of Pest Insects and Rodents, Institute of Zoology, Chinese Academy of Sciences, Beijing, 100101, China

<sup>5</sup> Department of Ecology and Evolutionary Biology, University of California, Irvine, CA, 92697, USA

<sup>6</sup> These authors contributed equally to this work.

<sup>†</sup> Corresponding author: [jje@uci.edu](mailto:jje@uci.edu) and [yiliao@scau.edu.cn](mailto:yiliao@scau.edu.cn).

# **Contents**

- 1.       Supplementary Notes**
- 2.       Supplementary Methods**
- 3.       Supplementary Figures**
- 4.       References**

# Supplementary Notes

## Supplementary Note 1: A practical guide to using SVGAP

### Section 1: Preparing the genome assemblies

#### Assembly quality requirement

The SVGAP pipeline was originally designed for genome assemblies of chromosome-level quality; however, it can also be effectively applied to contig-level assemblies. Theoretically, there are no specific quality requirements for the genome assembly, such as N50 or contig numbers. Nevertheless, we highly recommend that users include only chromosomes and long contigs when preparing the input genome assemblies. Currently, SVGAP is optimized for haploid genomes. If you have phased genomes with multiple haplotypes, the recommended approach is to treat each haplotype as an individual assembly.

#### Naming

To effectively utilize the SVGAP pipeline, users must adhere to specific naming guidelines for genome assemblies and their chromosomes/contigs. Here are some key points to keep in mind:

- 1) Assembly names do not need to include file extensions such as “.fasta” or “.fa”.
- 2) Avoid naming assemblies solely using numbers. For example, naming an assembly “9311” could potentially cause issues in certain situations. Instead, consider using “N9311”.
- 3) Do not name the query assembly in a way that completely matches or overlaps with the names of reference assemblies. For instance, if the reference name is “MH63”, it would be advisable to avoid naming any query assembly “H63” or “MH6”.
- 4) Chromosome names should follow the format “GenomeID.ChromosomeID”, such as “MH63.chr01”.

#### Splitting (Optional)

If you have an extreme size of plant genomes, such as maize and pepper, you may need to split the genome into small pieces, and then conduct the pairwise genome alignment. Here we provide a script called [0\\_SplitFa.pl](#) to facilitate this splitting work.

You just need to split the query one in the genome assemblies folder, here is an example:

```
$ perl O_SplitFa.pl ZH97 2000000 400000
```

This script will split a FASTA file into fixed-sized windows with a specified step size. The first parameter “ZH97” refers to the input FASTA file, the second parameter “2000000” indicates the size of the fixed-sized windows, and the third parameter “400000” denotes the step size for moving along the sequence.

## Section 2: a step-by-step tutorial

In this section, we used 49 rice genomes to demonstrate how to use SVGAP step by step:

### Constructing a working folder

To start the project, we first constructed a folder named rice:

```
$ mkdir rice  
$ mkdir rice/genome
```

### Moving working genomes to a subfolder

Once the working genomes have been named and filtered for short contigs, they are moved to the rice/genome subfolder. Please note that even if you split the query genomes, the rice/genome should only contain unsplit files.

### Step 0: Whole genome alignment

Currently, SVGAP supports alignment results from a variety of aligners, including Lastz, Last, MUMmer, minimap2, GSAAlign, and AnchorWave:

- 1) For small and medium-sized genomes (<500 Mb), we recommend utilizing Last and MUMmer.
- 2) For larger genomes, particularly in the case of plants, it is advisable to use minimap2 or AnchorWave instead.
- 3) If you opt for AnchorWave, please ensure that a dependable gene annotation GFF3 file is accessible for the reference genome and that there are no considerable rearrangements between the reference and query genomes.

In the case of our rice data, we opted to employ MUMmer4 (with default parameters) to conduct alignments between the reference genome (MH63) and each of the 48 query genomes.

Here's an example:

```
$ nucmer -t 4 --prefix=MH63vsBasm ./genome/MH63 ./genome/Basm
```

If you would like to use other alternative aligners, you may refer the following commands:

## minimap2

```
$ minimap2 -c --cs=long ./genome/MH63 ./genome/Basm > MH63vsBasm.chr.paf
```

## last

```
$ lastdb -P4 -uNEAR MH63dat ./genome/MH63
$ last-train -P4 --revsym -E0.05 -C2 MH63dat ./genome/Basm > MH63vsBasm.train
$ lastal -p MH63vsBasm.train MH63dat Basm > MH63vsBasm.maf
```

### Step 1: Converting the alignment output from a particular aligner to the AXT format alignment files

SVGAP incorporates a series of UCSC programs (e.g., *axtChain*, *ChainNet*, and *netSyntenic* developed by Jim Kent) to preprocess alignment files before identifying genomic variants. Hence, it is necessary to convert alignment files from other aligners with different formats, such as *.delta* from MUMmer4, *.paf* from minimap2, and *.maf* from LAST/LASTZ/AnchorWave, to the AXT alignment format (the *.axt* format is described in <http://genome.ucsc.edu/goldenPath/help/axt.html>) prior to processing them within the pipeline. To facilitate this conversion, SVGAP provides a script called *1\_Convert2Axt.pl* to handle this task. To find the detailed usage information about the *1\_Convert2Axt.pl* script, you can run:

```
$ perl 1_Convert2Axt.pl --help
```

In our case of rice data, we saved all *delta* alignment files produced by MUMmer4 to a single output folder *rice/alignment* and created a working folder *rice/chainnet/Target\_Ref*.

### Rearrange output files

```
$ mkdir ./rice/alignment
$ mkdir ./rice/chainnet
$ mv *.delta ./rice/alignment
```

### Run 1\_Convert2Axt.pl

```
$ perl 1_Convert2Axt.pl -ali mummer -input ./rice/alignment -wk ./rice/chainnet
```

Please note that if you split query genomes, you should provide a genome size file (tab-delimited form) named “genomeID.sizes” in the working folder where the .axt file will be saved before running *1\_Convert2Axt.pl*.

Here is an example of the genome size file:

```
MH63.chr01 23513712
MH63.chr02 32079331
```

After running the above commands, the resulting .axt files have been saved in the rice/chainnet folder and ready for further processing.

## Step 2: Constructing chains and nets from pairwise alignments

To accurately identify genetic variants between the reference and query genomes, the raw alignments (.axt file) must be processed for synteny, and paralogous regions need to be removed. This ensures that only the orthologous regions are retained for analysis. This task can be accomplished by employing a set of UCSC programs (find necessary information from [http://genomewiki.ucsc.edu/index.php/Whole\\_genome\\_alignmen\\_howto](http://genomewiki.ucsc.edu/index.php/Whole_genome_alignmen_howto)) that are sequentially invoked in the script *2\_ChainNetSyn.pl* within the SVGAP pipeline which will generate a filtered net file for syntenic alignments only and a single-coverage (on the reference genome) collection of pairwise alignments.

To find more detailed usage about *2\_ChainNetSyn.pl* script, run:

```
$ perl 2_ChainNetSyn.pl --help
```

In our case we run the command as follows:

```
$ perl 2_ChainNetSyn.pl --gd ../genome/ --ad chainnet/Target_Ref/ --lst ../genome/g.lst --sing
```

The filtered .net files for syntenic alignments will be saved in ./rice/synnet/ and the sing-coverage alignment .maf files will be saved in ./rice/singRaw/.

In most cases, especially when comparing two highly divergent genomes, it is recommended to utilize the additional script *3\_SynNetFilter.pl* to filter the syntenic alignment .net files. This script enhances precision while slightly reducing sensitivity, resulting in improved accuracy for such comparisons.

To see how to use the *3\_SynNetFilter.pl* script, run:

```
$ perl 3_SynNetFilter.pl --help
```

We run the following command for the rice test data:

```
$ perl 3_SynNetFilter.pl --synnet synnet/ --gd ../genome/ --chain chainnet/Target_Ref
```

The final filtered *.net* files for syntenic alignments are saved in *./rice/CleanSynNet/* which will be used for SV calling and the cleaned single-coverage alignment *.maf* files are saved in *./rice/filtered\_sing/* which will be used for SNP calling and SV genotyping.

### Step 3: Identification of structural variants (SVs)

The SVGAP pipeline infers SVs directly from the final filtered *net* files (the *net* format is described in <http://genome.ucsc.edu/goldenPath/help/net.html>). Currently, five common SV types between pairwise genome comparisons can be routinely identified using the script *4\_PairwiseSV.pl* within the SVGAP pipeline, including insertions, deletions, tandem duplications, inversions, and translations. Additionally, local highly divergent regions (i.e. with gaps on both target and query sequences) are defined as a complex type.

To see how to use the *4\_PairwiseSV.pl* script, you can run:

```
$ perl 4_PairwiseSV.pl --help
```

Below is the command we run for the rice data:

```
$ perl 4_PairwiseSV.pl --syndir ./rice/CleanSynNet --gd ./rice/genome/ --outdir ./rice/CleanSV
```

The detected SV calls are stored in the directory *./rice/CleanSV/*. For each pairwise comparison, two sets of SVs are reported, with coordinates that correspond to both the reference and query genomes.

### Step 4: Merge and remove redundant SV calls

Since SVs are identified between the reference genome and each of the query genomes, it is essential to merge the SVs from all comparisons. This merging process aims to generate a non-redundant dataset of SV calls that encompasses all the comparisons. By doing so, each SV can be further genotyped across all individuals involved in the study, enabling the construction of the pan-genome. The script *5\_Combined.pl* within the SVGAP pipeline can work for this task.

To see how to use the *5\_Combined.pl* script, you can run:

```
$ perl 5_Combined.pl --help
```

We run the following command to generate the non-redundant SV calls for all 48 pairwise companions.

```
$ perl 5_Combined.pl --sv ./rice/CleanSV --refname MH63
```

Executing the above command will generate a folder called ./rice/CombinedSV/. Within this folder, five types of non-redundant SVs will be generated. It is worth mentioning that the *insertions* and *deletions* have been categorized into two classes according to their lengths; specifically, events that are equal to or longer than 50 bp ( $\geq 50$  bp) are in one class, while the rest are in another.

### **Step 5: Re-genotyping the merged non-redundant SV calls across all samples based on pairwise (single coverage) alignments**

Although the above step has provided preliminary genotyping information, the information still lacks completeness. For instance, the detection of SVs was limited to specific individuals, leaving uncertainty as to whether the absence of these SVs in other individuals is genuine or merely a result of inaccurate sequencing and assembling, or sequence loss in those regions. Therefore, additional methods are necessary to achieve complete genotyping information for all SVs across all samples. The SVGAP utilizes pairwise single coverage alignment files to enhance the genotyping of each SV by extracting and comparing the local alignment in order to determine the SV status in each sample. The scripts [6\\_DELGenotyping.pl](#) and [7\\_INSGenotyping.pl](#) within the SVGAP pipeline are utilized to perform genotyping for deletions and insertions (with a length greater than or equal to 50 base pairs), respectively.

### **Genotyping Deletions ( $\geq 50$ bp)**

To see how to use the 6\_DELGenotyping.pl script, you can run:

```
$ perl 6_DELGenotyping.pl --help
```

We run the tested rice data with the following command:

```
$ perl 6_DELGenotyping.pl --del ./rice/CombinedSV/All.DELs.50bplarge.bed.combined.sorted.txt \
--singmaf ./rice/filtered_sing/ --refseq ./rice/genome/MH63 --refsize ./rice/genome/MH63.sizes \
--refname MH63 -t 12
```

This step will result in a *vcf* file named All.DELs.50bplarge.bed.combined.sorted.txt.vcf in the folder where the All.DELs.50bplarge.bed.combined.sorted.txt file is kept.



## Genotyping Insertions (≥50 bp)

To see how to use the *7\_INSGenotyping.pl* script, you can run:

```
$ perl 7_INSGenotyping.pl --help
```

We run the tested rice data with the following command:

```
$ perl 7_INSGenotyping.pl --ins ./rice/CombinedSV/All.INTs.50bplarge.bed.combined.sorted.txt \  
--query ../genome/ -sing filtered_sing/ --stretcher ../../SV-GAPS/pub/Stretcher/ --refseq ../genome/MH63 \  
--refsize ../genome/MH63.sizes --refname MH63
```

This step will result in a *vcf* file named All.INTs.50bplarge.bed.combined.sorted.txt.vcf in the folder where the All.INTs.50bplarge.bed.combined.sorted.txt file is kept.

## Genotyping InDels

Small deletions and insertions (<50 bp) are separately genotyped with *8\_InDelGenotyping.pl*.

To see how to use the *8\_InDelGenotyping.pl* script, you can run:

```
$ perl 8_InDelGenotyping.pl --help
```

Genotyping small deletions in our case:

```
$ perl 8_InDelGenotyping.pl --indel ./rice/CombinedSV/All.DELs.50bpsmall.bed.combined.sorted.txt \  
--indeltype del --refseq ./rice/genome/MH63 --refsize ./rice/genome/MH63.sizes --refname MH63 \  
--query ./rice/genome/ --singmaf ./rice/filtered_sing/ --outdir MH63_del --t 12
```

This step will result in a *vcf* file named All.DELs.50bpsmall.bed.combined.sorted.txt.vcf in the folder where the All.DELs.50bpsmall.bed.combined.sorted.txt file is kept.

Genotyping small insertions in our case:

```
$ perl 8_InDelGenotyping.pl --indel ./rice/CombinedSV/All.INTs.50bpsmall.bed.combined.sorted.txt \  
--indeltype ins --refseq ./rice/genome/MH63 --refsize ./rice/genome/MH63.sizes --refname MH63 \  
--query ./rice/genome/ --singmaf ./rice/filtered_sing/ --outdir MH63_ins --t 12
```

This step will result in a *vcf* file named All.INTs.50bpsmall.bed.combined.sorted.txt.vcf in the folder where the All.INTs.50bpsmall.bed.combined.sorted.txt file is kept.

## Genotyping SNPs

SVGAP also offers a script called *9\_SNPgenotyping.pl*, which allows for the direct calling of SNPs

from the single-coverage *maf* files. It generates a *vcf* file that includes all samples and their respective SNP information.

To see how to use the *9\_SNPgenotyping.pl* script, you can run:

```
$ perl 9_SNPgenotyping.pl --help
```

The command we used for the test data is:

```
$ perl 9_SNPgenotyping.pl --refseq ./rice/genome/MH63 --refsize ./rice/genome/MH63.sizes \
--refname MH63 --singmaf ../rice/filtered_sing/ --outdir SNPVCf --t 8
```

The SNP genotyped file All.SNP.vcf will report in the SNPVCF folder.

### Step 6: Annotating structural variants

When we acquire precise catalogs of structural variants, it is crucial to elucidate the potential mechanisms driving the formation of SVs and their functional impacts. We offer a script called [\*10\\_SVannotation.pl\*](#) to infer the primary common factors in plant genomes, such as tandem duplication, inverted tandem duplication, transposon elements (TEs) insertion, nest TEs insertion, and complete/partial gene duplication, that may be involved in SV formation.

To see how to use the *10\_SVannotation.pl* script, you can run:

```
$ perl 10_SVannotation.pl --help
```

The command we used for the test data is:

```
$ perl 10_SVannotation.pl --ref ./rice/genome/MH63 --vcf All.INTs.50bplarge.bed.combined.sorted.txt.vcf \
--svtype INS --te TE.EDTA.lab.fa --cds MSU7.cds.fa --out INS.annotated.vcf
$ perl 10_SVannotation.pl --ref ./rice/genome/MH63 --vcf All.DELs.50bplarge.bed.combined.sorted.txt.vcf \
--svtype DEL --te TE.EDTA.lab.fa --cds MSU7.cds.fa --out INS.annotated.vcf
```

The annotated *.vcf* file(s) will report in the current folder.

## Supplementary Note 2: Evaluation of execution time and memory usage for different genome aligners

### Installation of genome aligners

We selected 14 genome aligners to evaluate their execution time and memory usage for plant genome alignment, including minimap2, AnchorWave, MUMmer4, Blasr, BWA-MEN, GraphMap, Last, Lastz, MECAT2, Ngmlr, Pbmm2, GSAIalign, Unimap and winnowmap. We installed most of them through [conda](#) with one exception.

## Thirteen of them can be installed with conda

```
$ conda install -c bioconda minimap2
$ conda install -c bioconda -c conda-forge anchorwave
$ conda install bioconda::mummer4
$ conda install bioconda::blasr
$ conda install -c bioconda bwa
$ conda install bioconda::graphmap
$ conda install bioconda::last
$ conda install bioconda::lastz
$ conda install ngmlr -c bioconda
$ conda install -c bioconda pbmm2
$ conda install bioconda::gsalign
$ conda install bioconda::unimap
$ conda install bioconda::winnowmap
```

*MECAT2 was installed from GitHub :*

```
$ git clone https://github.com/xiaochuanle/MECAT2.git
$ cd MECAT2
$ make
```

### Testing for genome alignment

The command we used for the test data is:

#### 1. *minimap2 (version 2.22-r1101)*

## (2min 42s, 7.64 GB)

```
$ time minimap2 -x asm5 ~/MH63RS3.fasta ~/ZS97RS3.fasta > map.paf
```

#### 2. *AnchorWave (v1.2.3)*

*## Extract CDS (0.20 min, 959.61 Mb)*

```
$ time anchorwave gff2seq -i MH63RS3.gff3 -r MH63RS3.fasta -o cds.fa
```

*## Align CDS to the reference genome (3.20 min, 12.85 Gb)*

```
$ time minimap2 -x splice -t 10 -k 12 -a -p 0.4 -N 20 MH63RS3.fasta cds.fa > ref.sam
```

*## Align CDS to the query genome (3.53 min, 13.47 Gb)*

```
$ time minimap2 -x splice -t 10 -k 12 -a -p 0.4 -N 20 S97RS3.fasta cds.fa > query.sam
```

*## Perform genome alignment (78.52 min, 17.13 Gb)*

```
$ time anchorwave genoAli -i MH63RS3.gff3 -as cds.fa -r MH63RS3.fasta -a query.sam -ar ref.sam \
-s ZS97RS3.fasta -n SZ.anchors -o anchorwave.maf -f anchorwave.f.maf > SZ.log
```

### **3. MUMmer4 (4.0.0rc1)**

The command we used for the test data is:

*## Alignment (103.85 min, 4.65 Gb)*

```
$ time nucmer -p out MH63RS3.fasta ZS97RS3.fasta
```

### **4. Blasr (5.3.5-5.3.5)**

*Blasr* is used to align PacBio reads to the reference genome, but it keeps throwing errors during usage, and the cause is currently unclear.

The command we used for the test data is:

```
$ time blasr ZS97RS3.fasta MH63RS3.fasta --out test.out -m 5 --header --nproc 50
```

We tried modifying the parameters, but the errors persist:

```
$ time blasr ZS97RS3.fasta MH63RS3.fasta --out test.out --maxMatch 10000000000 --maxLCPLength 10000000000
```

### **5. BWA-MEM (0.7.17-r1188)**

The command we used for the test data is:

*## Construct genome index (4.22 min, 569.72 Gb)*

```
$ time bwa index MH63RS3.fasta
```

*## Alignment (time-consuming, 10.69 Gb)*

```
$ time bwa mem MH63RS3.fasta ZS97RS3.fasta > output.sam
```

## **6. GraphMap (v0.6.3)**

*GraphMap* is used to align Nanopore reads to the reference genome. The command we used for the test data is:

```
$ time graphmap2 align -r MH63RS3.fasta -d ZS97RS3.fasta -o graphmap_out.sam
```

Unfortunately, the result cannot be output due to our limited memory.

## **7. Last (1584)**

The command we used for the test data is:

*### Simple version*

*## Construct reference genome database (2.07 min, 1.89 Gb)*

```
$ time lastdb MHdb MH63RS3.fasta
```

*## Alignment (97.40 min, 3.44 Gb)*

```
$ time lastal MHdb ZS97RS3.fasta > myalns.maf
```

*## Make a dotplot (64.25 min, 118.68 Gb)*

```
$ time last-dotplot myalns.maf
```

*### Detailed version*

*## Construct reference genome database (0.48 min, 1.53 GB)*

```
$ time lastdb -P8 -c -uRY128 MH63db MH63RS3.fasta
```

*## Training (0.07 min, 651 Mb)*

```
$ time last-train -P8 --revsym -C2 MH63db ZS97RS3.fasta > zs.train
```

*## Mang to one alignment (1.17 min, 13.99 GB)*

```
$ time lastal -P8 -D1e9 -C2 --split-f=MAF+ -p zs.train MH63db ZS97RS3.fasta > many-to-one.maf
```

## One to one filtration (0.40 min, 2.31 GB)

```
$ time last-split -r many-to-one.maf > one-to-one.maf
```

## Make a dotplot (0.20 min, 98.39 Mb)

```
$ time last-dotplot one-to-one.maf
```

## 8. Lastz (1.04.22)

Lastz supports pairwise alignment of one chromosome to another, so we used a looping method for whole genome alignment:

## Extract each chromosome to a new FASTA file

```
$ awk '/^>/ {if (f) close(f); f = "MH63" substr($0, 2,5) ".fa"; print > f} !/^>/ {print > f}' MH63RS3.fasta  
$ awk '/^>/ {if (f) close(f); f = "ZS" substr($0, 2,5) ".fa"; print > f} !/^>/ {print > f}' ZS97RS3.fasta
```

## “Looping” alignment (8.47 min, 162.79 Mb)

```
$ for file in ZS*; do name="${file:2:5}"; lastz "MH63${name}.fa" "${file}" --notransition --step=20 --nogapped \\\n--format=maf > "${name}.maf"; done
```

## 9. MECAT2 (2.0.0)

MECAT2 is used to align SMRT reads to the reference genome. We used 80 threads because the execution was too slow:

## Alignment (45.47 min, 21.28 Gb)

```
$ time mecat2map -num_threads 80 ZS97RS3.fasta MH63RS3.fasta > result.m4
```

## 10. Ngmlr (0.2.7)

Ngmlr is also used to align SMRT reads to the reference genome.

The command we used for the test data is:

## Alignment (time-consuming)

```
$ time ngmlr -r ~/MH63RS3.fasta -q ~/ZS97RS3.fasta -o test.sam
```

### 11. *Pbmm2* (1.14.99)

*Pbmm2* is also used to align SMRT reads to the reference genome. We also used 80 threads because the execution was too slow:

## Construct genome index (0.10 min, 2.39 Gb)

```
$ time pbmm2 index MH63RS3.fasta ref.mmi
```

## Alignment (57.88 min, 438.5 Gb)

```
$ time pbmm2 align ref.mmi ZS97RS3.fasta ref.movie.bam -j 80
```

### 12. *GSA*Align (v1.0.22)

The command we used for the test data is:

## Alignment (5.45 min, 2.46 Gb)

```
$ time GSAAlign -r MH63RS3.fasta -q ZS97RS3.fasta -o output
```

### 13. *Unimap* (0.1-r41)

The command we used for the test data is:

### Index-free genome alignment (2.28 min, 9.28 Gb)

```
$ time unimap -b24 MH63RS3.fasta ZS97RS3.fasta > out.paf
```

### Index-depended genome alignment

## Construct genome index (0.67 min, 7.31 Gb)

```
$ time unimap -b24 -d out.umi ~/MH63RS3.fasta
```

## Alignment (2.23 min, 6.21 Gb)

```
$ time unimap -c out.umi ~/ZS97RS3.fasta > out1.paf
```

### 14. *winnowmap* (2.03)

The command we used for the test data is:

## Meryl count (1.82 min, 2.21 Gb)

```
$ time meryl count k=19 output merylDB MH63RS3.fasta
```

*## Meryl print (0.12 min, 1.08 Gb)*

```
$ time meryl print greater-than distinct=0.9998 merylDB > repetitive_k19.txt
```

*## Alignment (6.45 min, 6.44 Gb)*

```
$ time winnowmap -W repetitive_k19.txt -x asm20 MH63RS3.fasta ZS97RS3.fasta > output.paf
```



# Supplementary Methods

## Approaches for benchmarking analysis using a phylogeny-based simulated dataset

To evaluate the extent to which sequence divergence impacts SV detection, we propose the following strategy for benchmark analysis. **First**, we selected eight divergent genomes, using one as the reference, from both the *Oryza* and *Solanum* phylogenies. The genomes exhibit varying levels of sequence divergence relative to the reference. **Second**, for each phylogeny-based genome set, we simulated SVs, including insertions and deletions, in all eight genomes using RSVSim <sup>1</sup>. **Third**, we identified SVs between the reference and the eight simulated genomes, as well as between the reference and the eight original genomes. **Fourth**, the latter SV set (background calls) was subtracted from the former to obtain the set of induced SVs. **Finally**, these remaining SV calls were compared against the simulated SV set to measure the recall, precision, and *F1*-score for each genome.

### Step 1: Simulating SVs in the genomes

We used the following script, run\_RSVSim.Rscript, to simulate SVs for each genome.

```
#!/usr/bin/env Rscript
library(argparser, quietly = TRUE)
## Create a parser
p <- arg_parser("RSVSim runner")
## Add command line arguments
p <- add_argument(p, "--genome", help = "genome_path", type = "character")
p <- add_argument(p, "--output", help = "output_path", type = "character")
p <- add_argument(p, "--delnum", help = "delnum", type = "character")
p <- add_argument(p, "--insnum", help = "insnum", type = "character")
## Parse the command line arguments
argv <- parse_args(p)
genome_file <- argv$genome
output_file <- argv$output
delnum <- as.numeric(argv$delnum)
insnum <- as.numeric(argv$insnum)
library("RSVSim")
## Setting SV size for DELs and INTs
delSizes = estimateSVSizes(n = delnum, minSize = 50, maxSize = 20000, default = "deletions", hist = TRUE)
intSizes = estimateSVSizes(n = insnum, minSize = 50, maxSize = 20000, default = "insertions", hist = TRUE)
## With additional breakpoint mutations
simulateSV(output = output_file, genome = genome_file, sizeDels = delSizes, dels = delnum, sizeIns = intSizes, ins = insnum,
```

```
percCopiedIns = 1)
```

For example, we used the following command to introduce 10,000 deletions and 10,000 insertions into the CJ14 genome.

```
$ Rscript run_RSVMsim.Rscripts --genome CJ14 --output CJ14_Simulated --delnum 10000 --insnum 10000
```

The output are files recording the variant's coordinates: **deletions.csv** and **insertions.csv**, and the simulated genome: **genome\_rearranged.fasta**, all have been saved in the folder CJ14\_Simulated.

Since the coordinates in deletions.csv and insertions.csv are based on the original genomes, they need to be lifted over to the simulated genomes for comparisons between the reference and the simulated genomes. The following commands and the script SimulationReformatSV.pl were used for this purpose.

```
$ cat deletions.csv | grep -v Name | cut -f1-5 > simulatedA.DELs.bed
$ cat insertions.csv | grep -v Name | cut -f1,5-8 > simulatedA.INTs.bed
$ cat simulatedA.DELs.bed simulatedA.INTs.bed | sort -k2,2 -k3,3n > simulatedA.All.variant.sort.bed
$ perl SimulationReformatSV.pl simulatedA.All.variant.sort.bed
```

This step will generate .bed files containing SV coordinates based on the simulated genomes.

## Step 2: Whole genome alignment

The generated simulated data can be used for benchmarking aligners and callers. Here we show how we benchmark callers with SVGAP, including Last, MUMmer4, minimap2 and AnchorWave.

```
### LAST
## Construct reference genome database
$ lastdb -P8 -uNEAR Adatabase ./genome/A
## Training
$ last-train -P8 -E0.05 --revsym -C2 Adatabase ./genome/B > AvsB.train
## Alignment
$ lastal --split-f=MAF -p AvsB.train Adatabase ./genome/B > AvsB.chr.maf

### MUMmer4
$ nucmer -t 4 --prefix=AvsB ./genome/A ./genome/B

### minimap2
$ minimap2 -t 2 -c --cs=long -x asm5 ./genome/A ./genome/B > AvsB.chr.paf

### AnchorWave
## Extract CDS
$ anchorwave gff2seq -i ./genome/A.gff -r ./genome/A -o ./genome/A.CDS.fa
## Align CDS to the query genome
$ minimap2 -x splice -t 10 -k 12 -a -p 0.4 -N 20 ./genome/B ./genome/A.CDS.fa > B.sam
## Align CDS to the reference genome
```

```
$ minimap2 -x splice -t 10 -k 12 -a -p 0.4 -N 20 ./genome/A ./genome/A.CDS.fa > A.sam
## Perform genome alignment
$ anchorwave genoAli -i ./genome/A.gff -as ./genome/A.CDS.fa -r ./genome/A -a B.sam -ar A.sam -s ./genome/B -n B.anchors \
-o B.maf -f AvsB.maf > B.log
```

### Step 3: Calling SVs

Using the alignment results from the four tested aligners, we applied the SVGAP pipeline to call SVs between each genome and the reference. The single-coverage alignment *.maf* files (saved in the singRaw directory) and the detected SV calls (stored in the CleanSV directory) were used for subsequent analysis.

We used the following command to calculate the alignable proportion of each genome relative to the reference.

```
$ grep -v '#' singRaw/A.simulatedB.sing.maf | awk -v term="simulatedB" '$2 ~ term' | \
awk '{sum += $4} END {print sum}' > AvssimulatedB.alignedLength

## Calculate the total genome size
$ seqkit faidx genome/simulated
$ awk '{sum += $2} END{print sum}' simulatedB.fai > simulatedB.totallength

$ echo "$(cat AvssimulatedB.alignedLength)/$(cat simulatedB.totallength)" | bc -l | \
awk -v prefix="mappingrate=" '{print prefix $0}'
```

### Step 4 : Benchmarking

We calculated recall, precision, and F1 score using the following formula:

```
Precision = TP / (TP + FP)
Recall = TP / (TP + FN)
F1 Score = (2 * Precision * Recall) / (Precision + Recall)
```

, where TP, FP, FN and TN are defined as following:

# TP (True positive) SVGAP predicted there is a SV and it actually has.

# FP (False positive): SVGAP predicted there is a SV but it actually hasn't.

# FN (False negative): SV-GAPS predicted there isn't a SV but it actually has.

#TN (True negative): SVGAP predicted there isn't a SV and it actually hasn't.

**Benchmark aligners for DEL (deletion) between genome A and simulated genome A'**

```

## Select DELs ≥ 50 bp
$ cat CleanSV/A.simulatedA.rawSV/AvssimulatedA.chain.filter.tnet.synnet.refiltered.synnet.out.indel.sort | grep -v "#" | awk
'$8>49 {print $5"\t"$6"\t"$6+$8}' > AvssimulatedA.DELs.bed
## Calculation for TP
$ bedtools intersect -wa -wb -a simulatedA.All.variant.sort.bed.del.bed -b AvssimulatedA.DELs.bed -f 0.8 -F 0.8 | cut -f1-3 |
sort | uniq | wc -l | awk -v prefix="TP=" '{print prefix $0}'
## Calculation for FP
$ bedtools intersect -b simulatedA.All.variant.sort.bed.del.bed -a AvssimulatedA.DELs.bed -f 0.8 -F 0.8 -v | sort | uniq | wc -l |
awk -v prefix="FP=" '{print prefix $0}'
## Calculation for FN
$ bedtools intersect -a simulatedA.All.variant.sort.bed.del.bed -b AvssimulatedA.DELs.bed -f 0.8 -F 0.8 -v | sort | uniq | wc -l |
awk -v prefix="FN=" '{print prefix $0}'

```

### **Benchmark aligners for INT (insertion) between genome A and simulated genome A'**

```

## Select INTs ≥ 50 bp
$ cat CleanSV/simulatedA.A.rawSV/AvssimulatedA.chain.filter.qnet.synnet.refiltered.synnet.out.indel.sort | awk '$8>49' \
> AvssimulatedA.INTs.bed
## Calculation for TP
$ bedtools intersect -wa -wb -a simulatedA.All.variant.sort.bed.ins.bed -b AvssimulatedA.INTs.bed -f 0.8 -F 0.8 | cut -f1-3 | \
sort | uniq | wc -l | awk -v prefix="TP=" '{print prefix $0}'
## Calculation for FP
$ bedtools intersect -b simulatedA.All.variant.sort.bed.ins.bed -a AvssimulatedA.INTs.bed -f 0.8 -F 0.8 -v | sort | uniq | wc -l | \
awk -v prefix="FP=" '{print prefix $0}'
## Calculation for FN
$ bedtools intersect -a simulatedA.All.variant.sort.bed.ins.bed -b AvssimulatedA.INTs.bed -f 0.8 -F 0.8 -v | sort | uniq | wc -l | \
awk -v prefix="FN=" '{print prefix $0}'

```

### **Benchmark aligners for DEL (deletion) between genome A and simulated genome B'**

```

## Select DELs ≥ 50 bp
$ awk '$8>49' CleanSV/A.B.rawSV/AvsB.chain.filter.tnet.synnet.refiltered.synnet.out.indel.sort > AvsB.DELs.bed
$ awk '$8>49' CleanSV/A.simulatedB.rawSV/AvssimulatedB.chain.filter.tnet.synnet.refiltered.synnet.out.indel.sort \
> AvssimulatedB.DELs.bed
## Extract the different DELs set between A vs. B and A vs. simulatedB
$ bedtools intersect -a AvssimulatedB.DELs.bed -b AvsB.DELs.bed -v -f 0.8 -F 0.8 | awk '{print $5"\t"$6"\t"$6+$8}' \
> simulatedBvsB.diff.DELs.bed
## Calculation for TP
$ bedtools intersect -wa -wb -a simulatedB.All.variant.sort.bed.del.bed -b simulatedBvsB.diff.DELs.bed -f 0.8 -F 0.8 | \
cut -f1-3 | sort | uniq | wc -l | awk -v prefix="TP=" '{print prefix $0}'
## Calculation for FP

```

```
$ bedtools intersect -b simulatedB.All.variant.sort.bed.del.bed -a simulatedBvsB.diff.DELs.bed -f 0.8 -F 0.8 -v | sort | uniq | \
wc -l | awk -v prefix="FP=" '{print prefix $0}'
## Calculation for FN
$ bedtools intersect -a simulatedB.All.variant.sort.bed.del.bed -b simulatedBvsB.diff.DELs.bed -f 0.8 -F 0.8 -v | sort | uniq | \
wc -l | awk -v prefix="FN=" '{print prefix $0}'
```

### **Benchmark aligners for INT (insertion) between genome A and simulated genome B'**

```
## Select INTs ≥50 bp
$ grep -v '#' CleanSV/B.A.rawSV/AvsB.chain.filter.qnet.synnet.refiltered.synnet.indel.sort | \
awk '$8>49 {print $5"\t"$6"\t"$6+$8"\t"$1"\t"$2"\t"$3}' > AvsB.INTs.bed
$ grep -v '#' CleanSV/simulatedB.A.rawSV/AvssimulatedB.chain.filter.qnet.synnet.refiltered.synnet.indel.sort | \
awk '$8>49 {print $5"\t"$6"\t"$6+$8"\t"$1"\t"$2"\t"$3}' > AvssimulatedB.INTs.bed
## Extract the different INTs set between A vs. B and A vs. simulatedB
$ bedtools intersect -a AvssimulatedB.INTs.bed -b AvsB.INTs.bed -v -f 0.8 -F 0.8 | cut -f4-6 > simulatedBvsB.diff.INTs.bed
## Calculation for TP
$ bedtools intersect -a simulatedB.All.variant.sort.bed.ins.bed -b simulatedBvsB.diff.INTs.bed -f 0.8 -F 0.8 -wa -wb | \
cut -f1-3 | sort | uniq | wc -l | awk -v prefix="TP=" '{print prefix $0}'
## Calculation for FP
$ bedtools intersect -b simulatedB.All.variant.sort.bed.ins.bed -a simulatedBvsB.diff.INTs.bed -f 0.8 -F 0.8 -v | sort | uniq | \
wc -l | awk -v prefix="FP=" '{print prefix $0}'
## Calculation for FN
$ bedtools intersect -a simulatedB.All.variant.sort.bed.ins.bed -b simulatedBvsB.diff.INTs.bed -f 0.8 -F 0.8 -v | sort | uniq | \
wc -l | awk -v prefix="FN=" '{print prefix $0}'
```

### **Step 5: Normalization**

We used the alignable proportion to normalize the metrics, ensuring that the precision, recall, and F1-score are calculated only from the alignable regions. These metrics were further adjusted using the genome alignment rate, as described below:

#  $average\_AlignmentRate = \frac{1}{N_{(The\ number\ of\ aligners)}}$

#  $adjustTP = TP / average\_AlignmentRate$

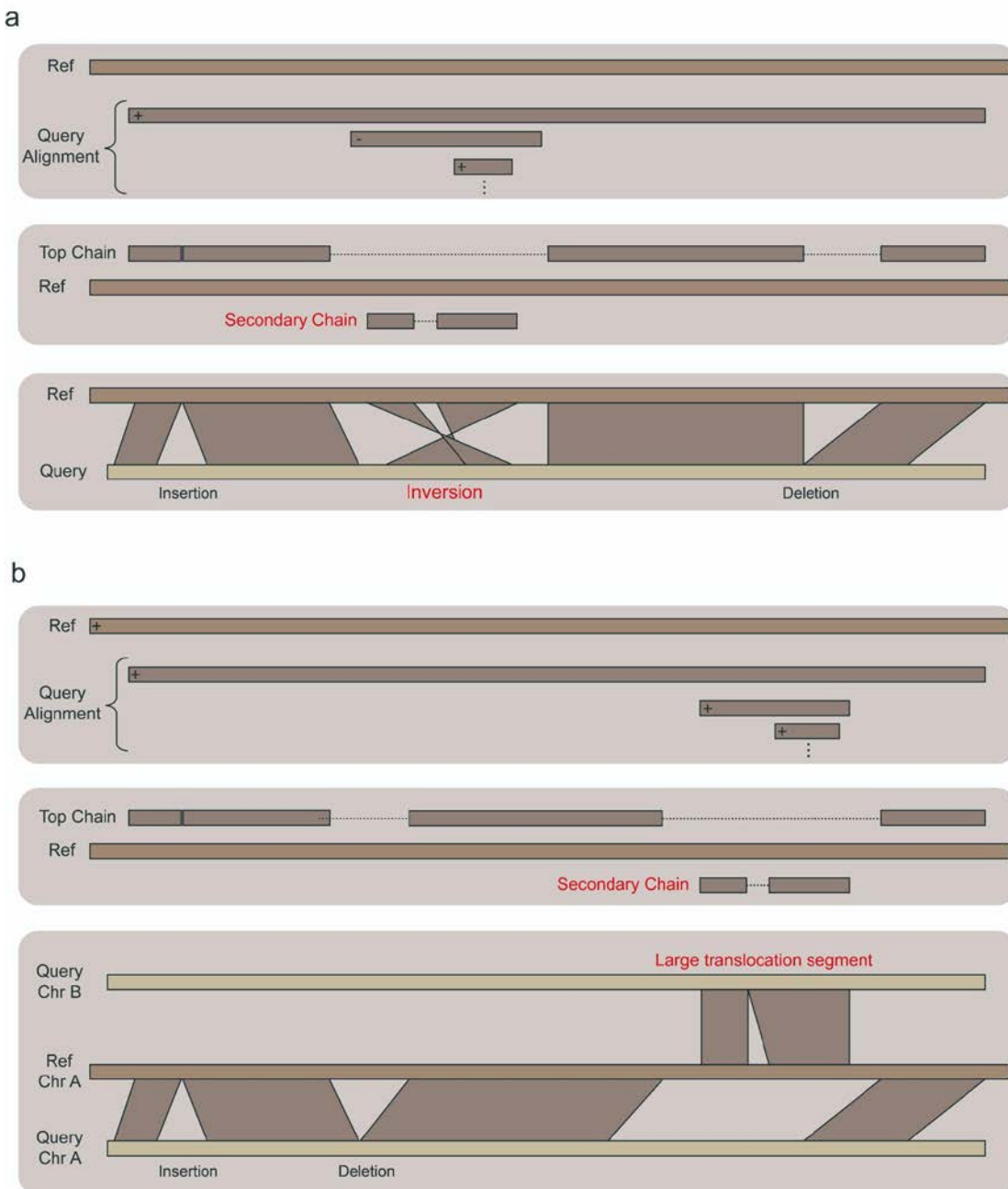
#  $adjustFP = FP * average\_AlignmentRate$

#  $adjustFN = N_{(SV\ number)} - adjustTP$

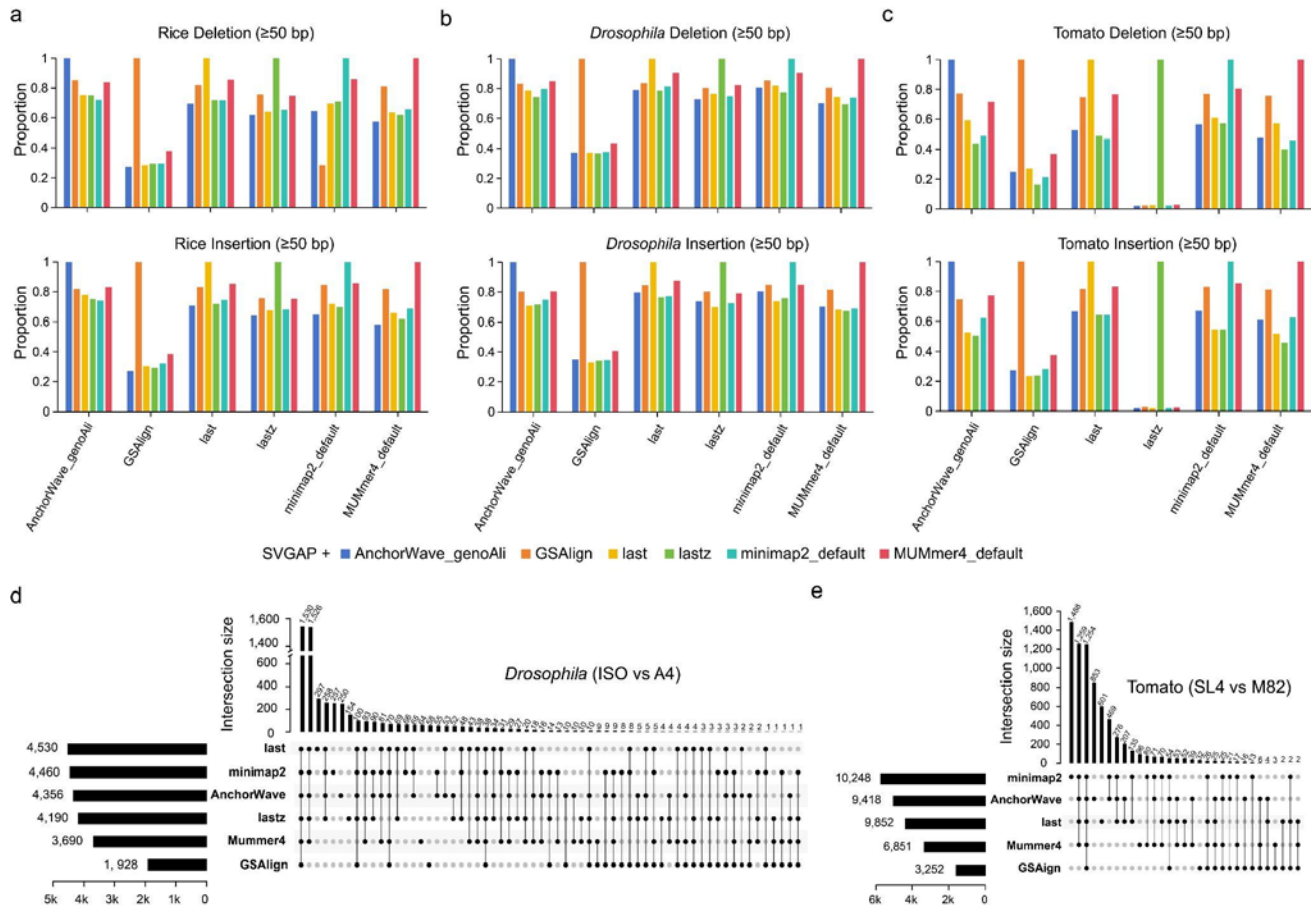
## **PGGB, Cactus, and AnchorWave for population-scale genome assemblies**

For PGGB, a variation graph was constructed using `pggb -i All.fa -o All -t 16 -p 90 -s 5k -V NIPP:1000000`, where `All.fa` contains all genome assemblies, `-p 90` sets the percent identity threshold to 90%, `-s 5k` sets the segment length to 5 kb, and `-V NIPP:1000000` anchors the graph to the NIPP genome. For Cactus, two alternative strategies were evaluated. The first strategy uses the integrated cactus-pangenome workflow: `cactus-pangenome js genome.pos.txt --outDir out --outName rice_pg --reference NIPP --vcf --giraffe --gfa --gbz --workDir $PWD --maxCores 24 --maxMemory 120G`, which builds the pangenome graph, infers variants, and produces GFA, GBZ, Giraffe, and VCF outputs in a single run. The second strategy adopts a modular approach: `cactus-minigraph js01 genome.pos.txt rice.gfa.gz --reference NIPP`, `cactus-graphmap js02 genome.pos.txt rice.gfa.gz rice.paf --reference NIPP --outputFasta rice.gfa.fa.gz`, `cactus-align js03 genome.pos.txt rice.paf rice.hal --pangenome --outVG --reference NIPP --maxCores 48 --maxMemory 250G --maxDisk 200G`, and `cactus-graphmap-join js04 --vg rice.vg --outDir ./ --outName rice-pg --reference NIPP --vcf --giraffe`, which decouples graph construction, mapping, alignment, and variant extraction for more flexible usage. For AnchorWave + TASSEL, genome alignments were conducted using `anchorwave genoAli -i NIPP.gff -as cds.fa -r NIPP -a ler.sam -ar ref.sam -s SIM1 -n anchors -o SIM1.maf -f SIM1.f.maf -t 24`, leveraging gene annotations and reference coding sequences. The resulting MAF files were converted into gVCF format using TASSEL's `run_pipeline.pl -Xmx100G -debug -MAFToGVCFPlugin -referenceFasta NIPP -mafFile xx.maf -sampleName xx -gvcfOutput SIM1.gvcf -fillGaps false`. All pipelines were benchmarked using the same simulated data to assess their recall, accuracy, error rate and missing in detecting insertions and deletions, with results compared directly to SVGAP.

## Supplementary Figures

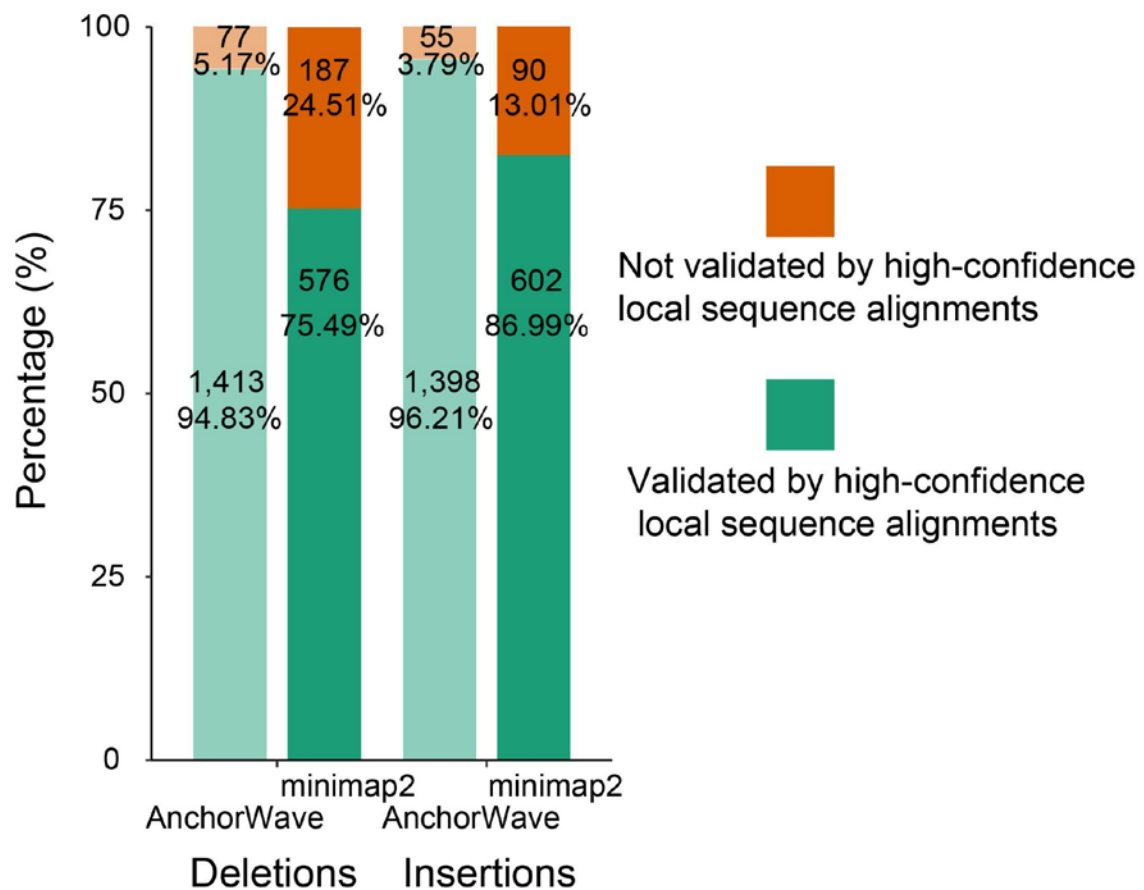


**Supplementary Figure 1: SVGAP facilitates structural variant (SV) identification in inversion and large translocation segments.** In most cases, SVs were detected from the sequence alignments in the top chain. However, some regions exhibited large inversions (**a**) or translocations (**b**), which are more likely to appear as lower (i.e., secondary) chains. These regions are rearranged genomic areas that remain syntenic and orthologous, indicating that the SVs within them should be identified. SVGAP is capable of identifying SVs in these regions and constructing a comprehensive SV dataset.

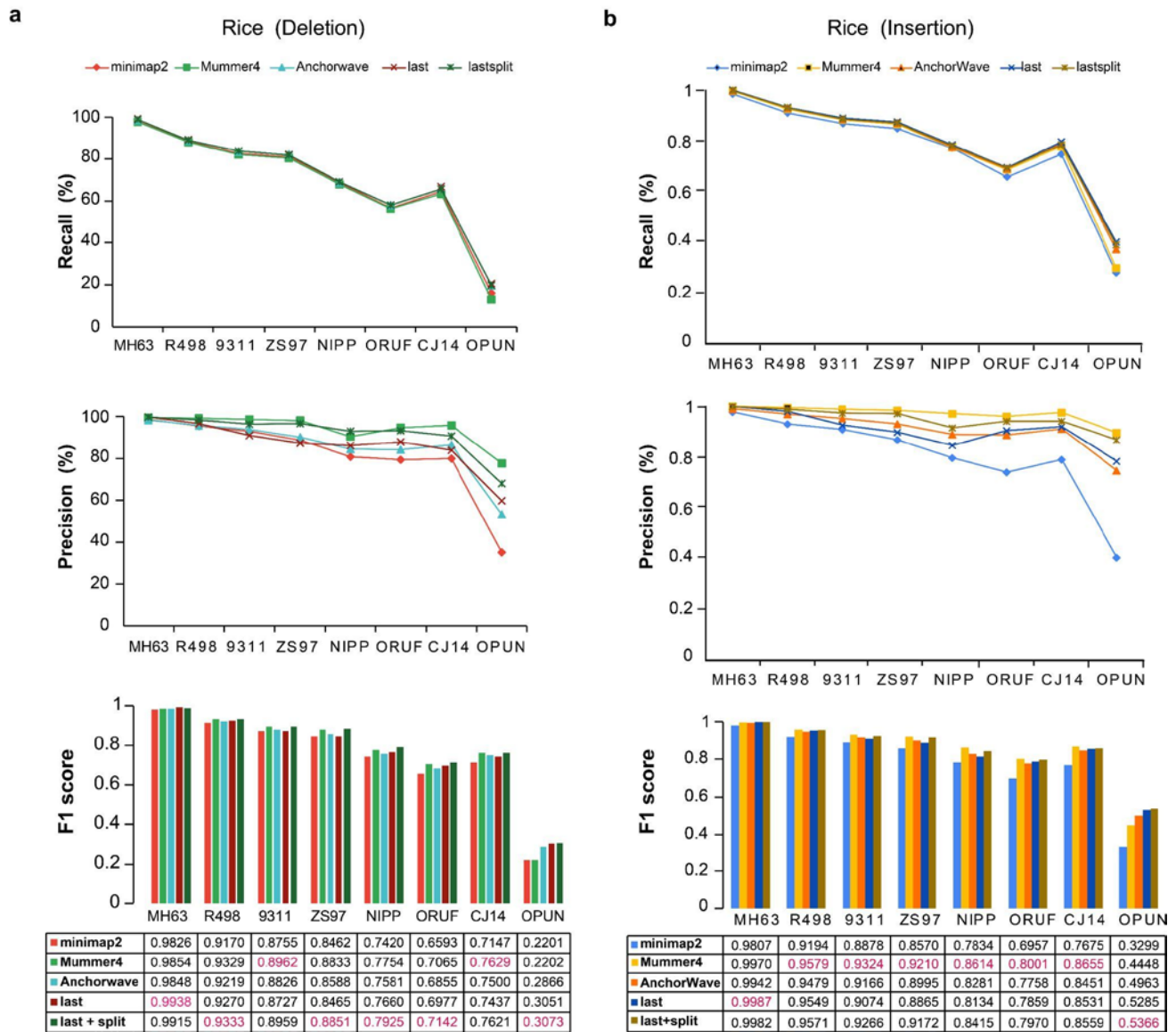


**Supplementary Figure 2: Comparison of SVGAP with different aligners for identifying SVs between real genomes in rice, *Drosophila*, and tomato.** **a.** Pairwise overlap of SVs (deletions and insertions) identified by SVGAP using alignment results from different aligners between rice genome assemblies: MH63 and ZS97. **b.** Pairwise overlap of SVs identified by SVGAP using alignment results from different aligners between two *Drosophila* genome assemblies: iso-1 and A4. **c.** Pairwise overlap of SVs identified by SVGAP using alignment results from different aligners between two tomato genome assemblies: SL4 and M82. **d.** An upset plot showing the analysis of shared calls between different aligners on two *Drosophila* data. **e.** An upset plot showing the analysis of shared calls between different aligners on tomato data. The exact numbers are available in [Supplementary Table 4](#).

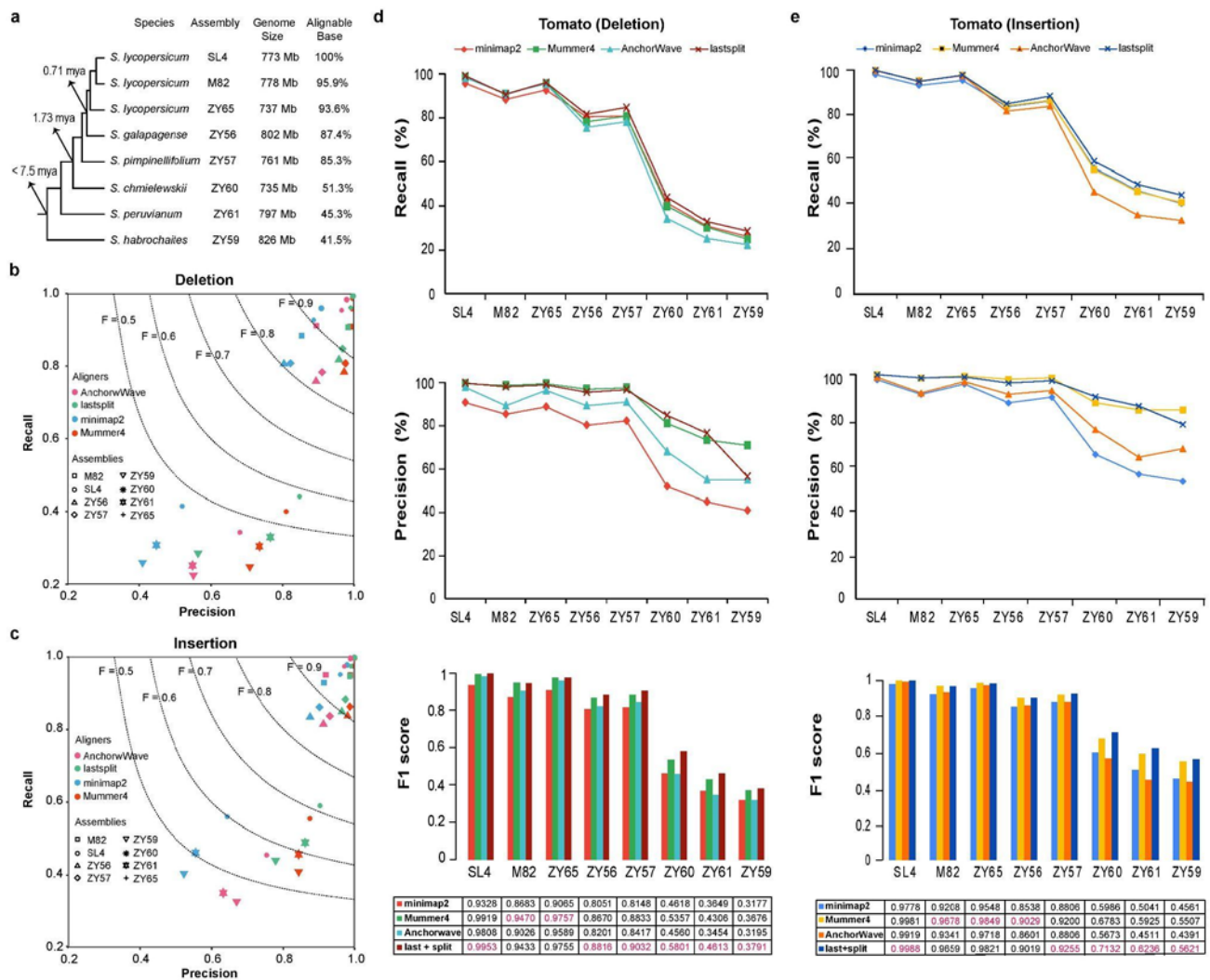




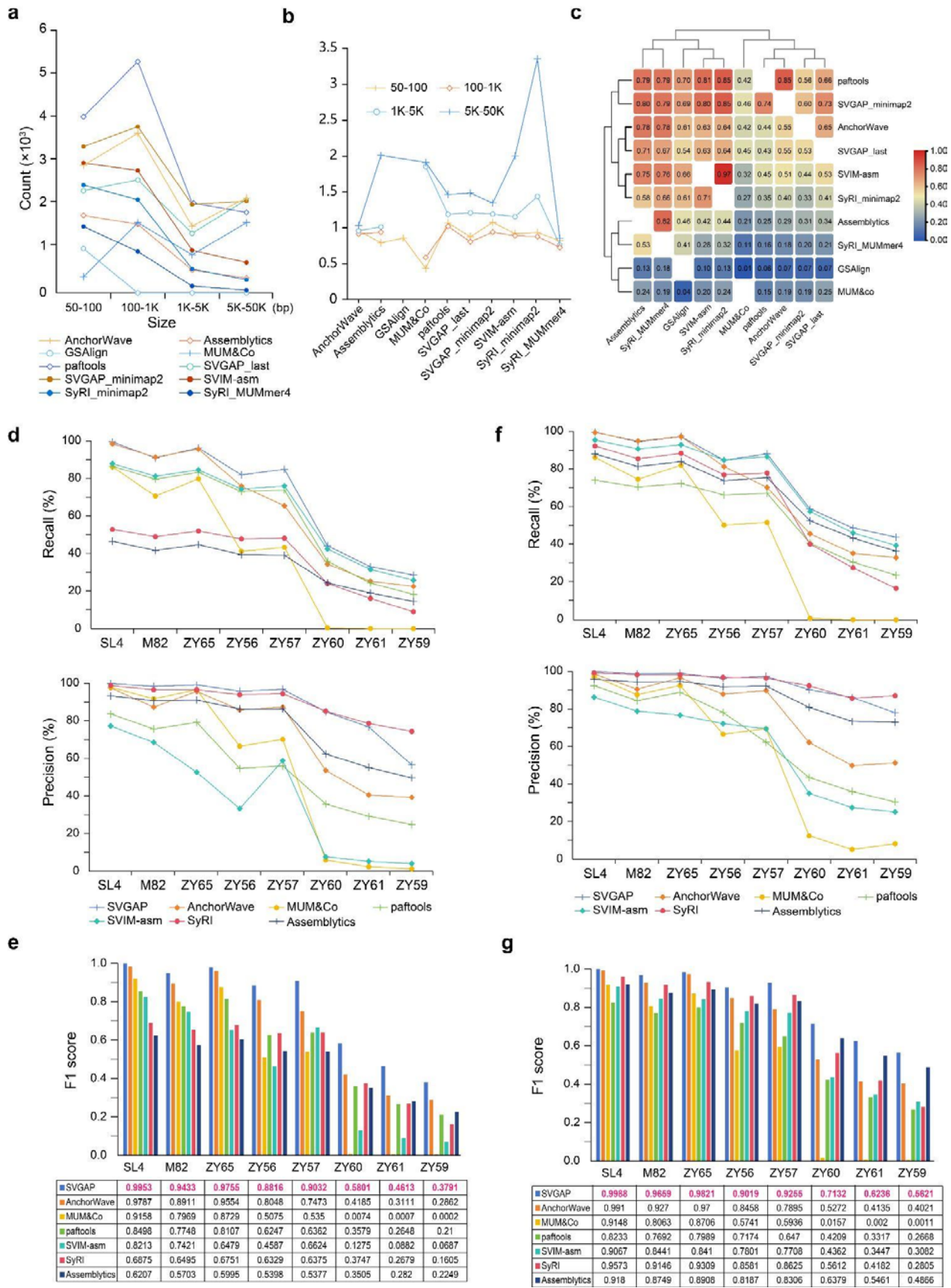
**Supplementary Figure 3: Verification of aligner-specific SVs by revisiting the local alignments generated by each corresponding aligner.** Deletions and insertions identified by SVGAP in combination with AnchorWave and minimap2 were confirmed through inspection of their respective local alignments.



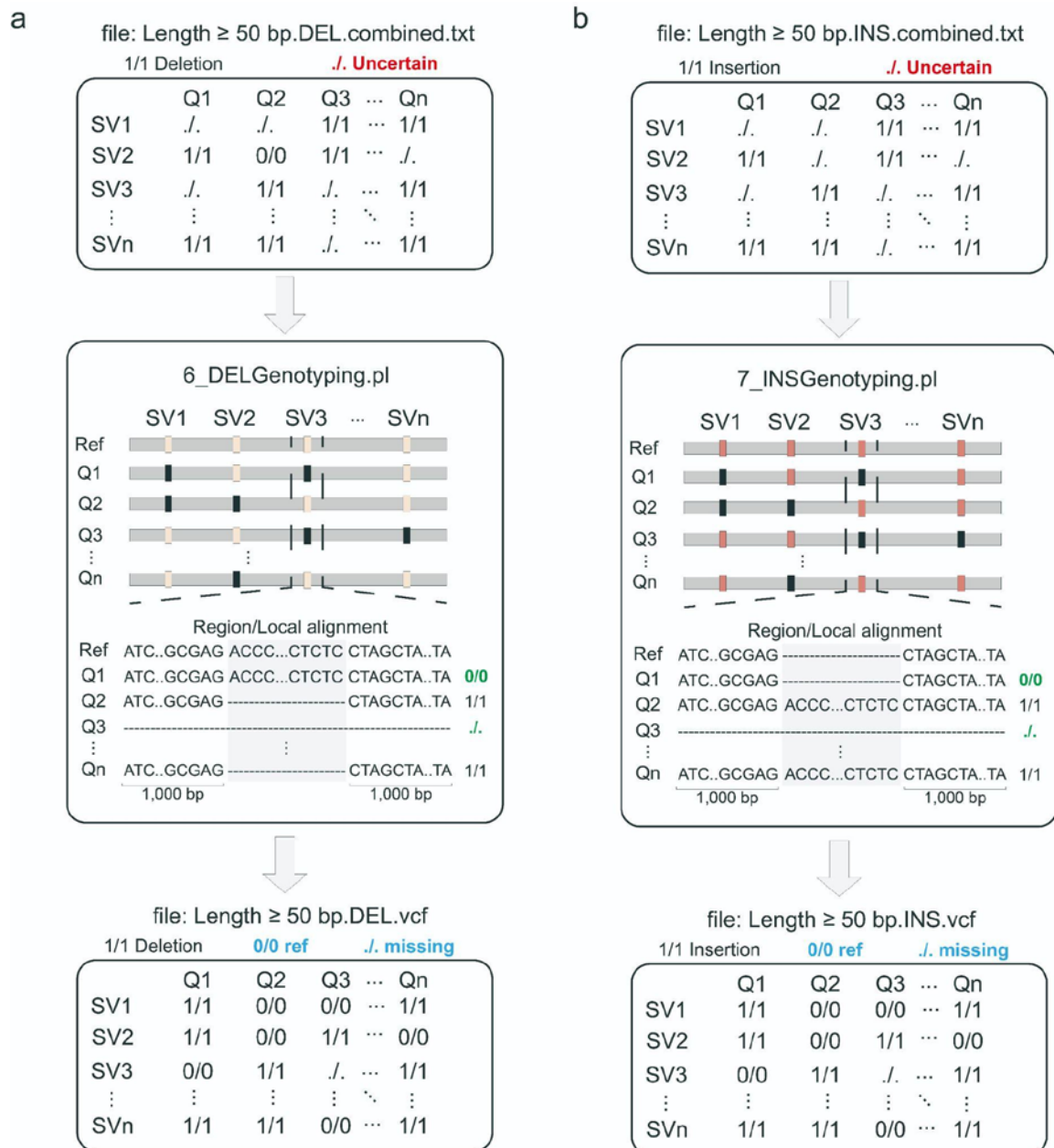
**Supplementary Figure 4: Benchmarking SVGAP with different aligners for SV calling using *Oryza* phylogeny-based simulation data. a. Recall, precision, and F1 score for deletions. b. Recall, precision, and F1 score for insertions. The exact numbers are available in [Supplementary Table 6](#).**



**Supplementary Figure 5: Benchmarking SVGAP with different aligners for SV calling using *Solanum* phylogeny-based simulation data.** **a.** The *Solanum* genomes, phylogenetic relationship and sequence divergence used for benchmark analysis. **b.** Recall, precision, and F1 score for deletions. **c.** Recall, precision, and F1 score for insertions. **d** and **e** showing the trend of each metric changes as sequence divergence. The exact numbers are available in [Supplementary Table 6](#).

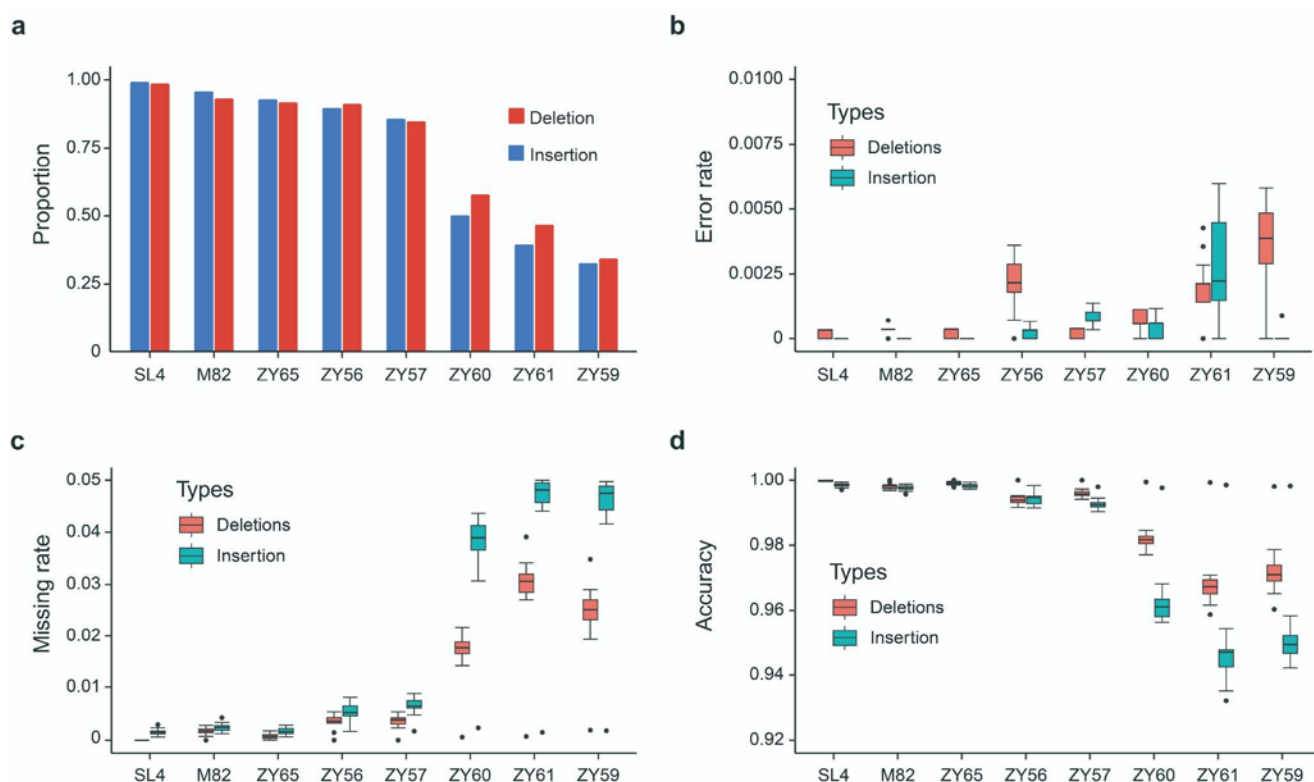


**Supplementary Figure 6: Comparison of SVGAP with other methods using benchmark analysis based on simulated data from tomato and *Solanum* phylogeny.** **a.** Number of SV calls reported by different methods across various length ranges when comparing the two tomato genomes, SL4 and M82. **b.** Ratio of deletions to insertions reported by different methods based on the two tomato genomes. **c.** Overlap of SVs in pairwise comparisons among different methods. **d.** Recall and precision for the benchmark analysis of deletions using simulated data from *Solanum* phylogeny. **e.** F1 score for deletions. **f.** Recall and precision for insertions. **g.** F1 score for insertions.

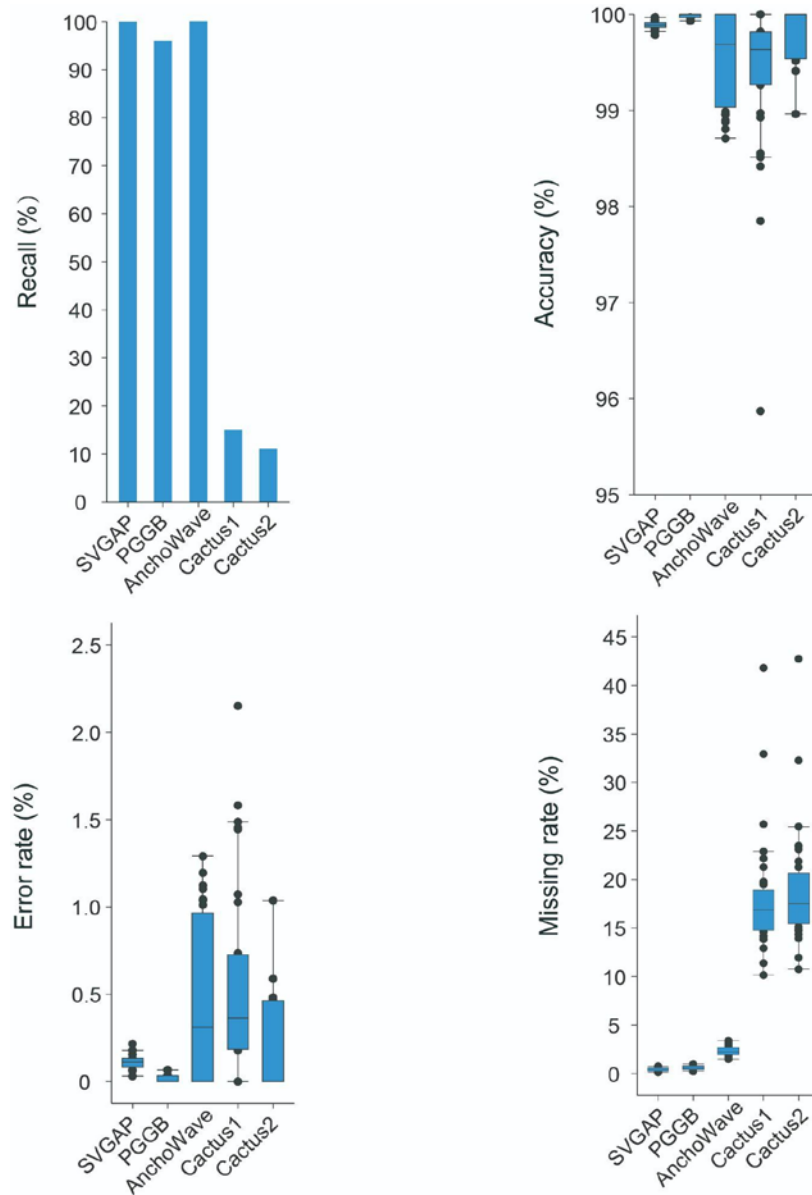


**Supplementary Figure 7: The diagram illustrates the strategy of population genotyping for deletions and insertions.** The genotyping step takes a pre-combined SV dataset as input and re-genotypes each SV call across all samples by extracting local alignments around the SV site from the corresponding pairwise single-coverage genome alignments. For each sample and SV, SVGAP reports three genotypes: 1/1 (alternative type), 0/0 (reference type), and ./ (missing or unable to determine).





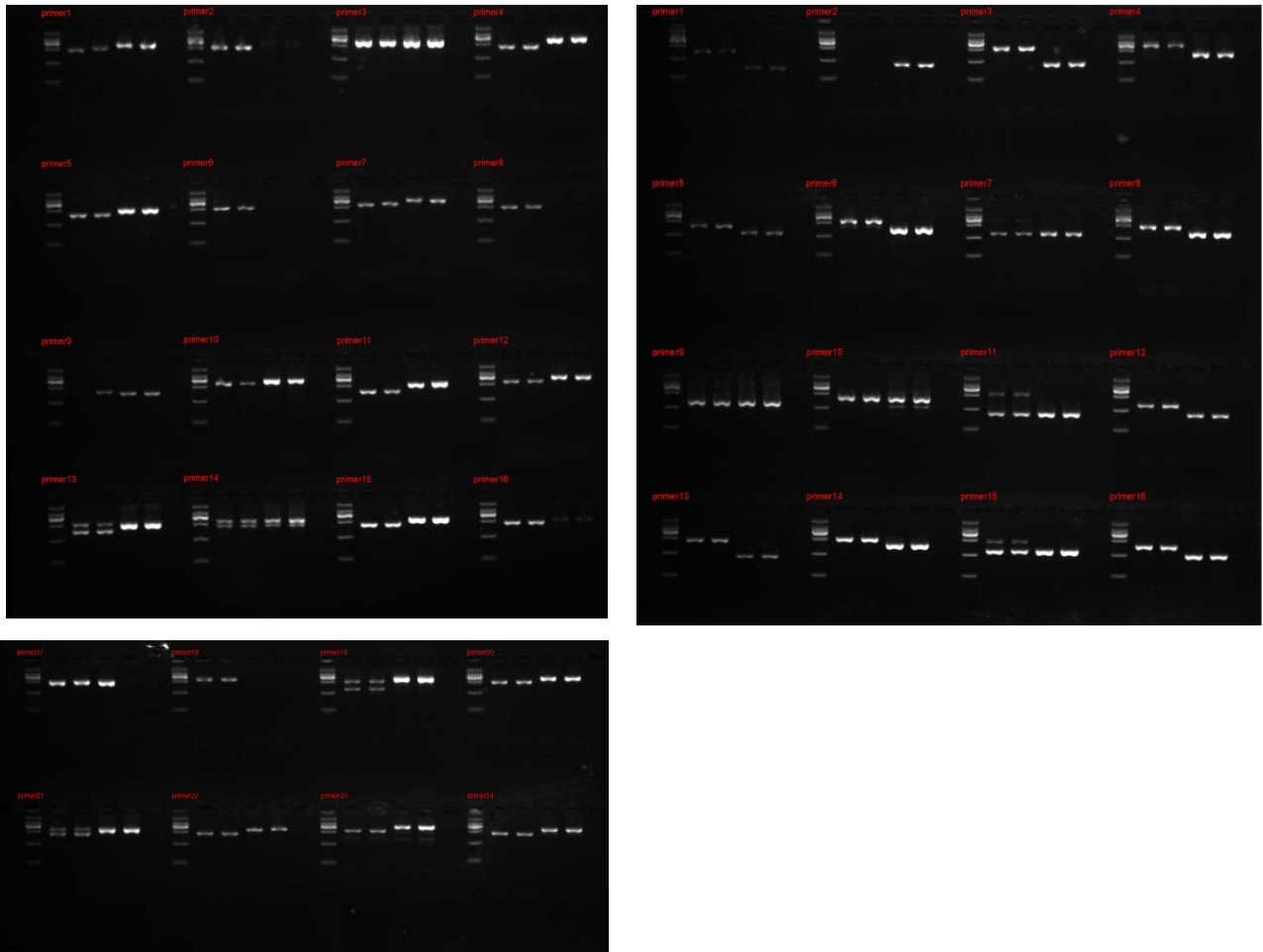
**Supplementary Figure 8: Benchmark analysis of the merging and genotyping functions in SVGAP with tomato data.** **a**, Proportion of retrieved simulated calls in the merging step by comparing the reference SL4 and eight other *Solanum* genomes of varying sequence divergence. **b**, Error rate of the genotyping step with population-scale simulating data based on *Solanum* phylogeny. **c**, The missing rate of the genotyping step across genomes of varying sequence divergence. **d**, The accuracy of the genotyping step across genomes of varying sequence divergence.



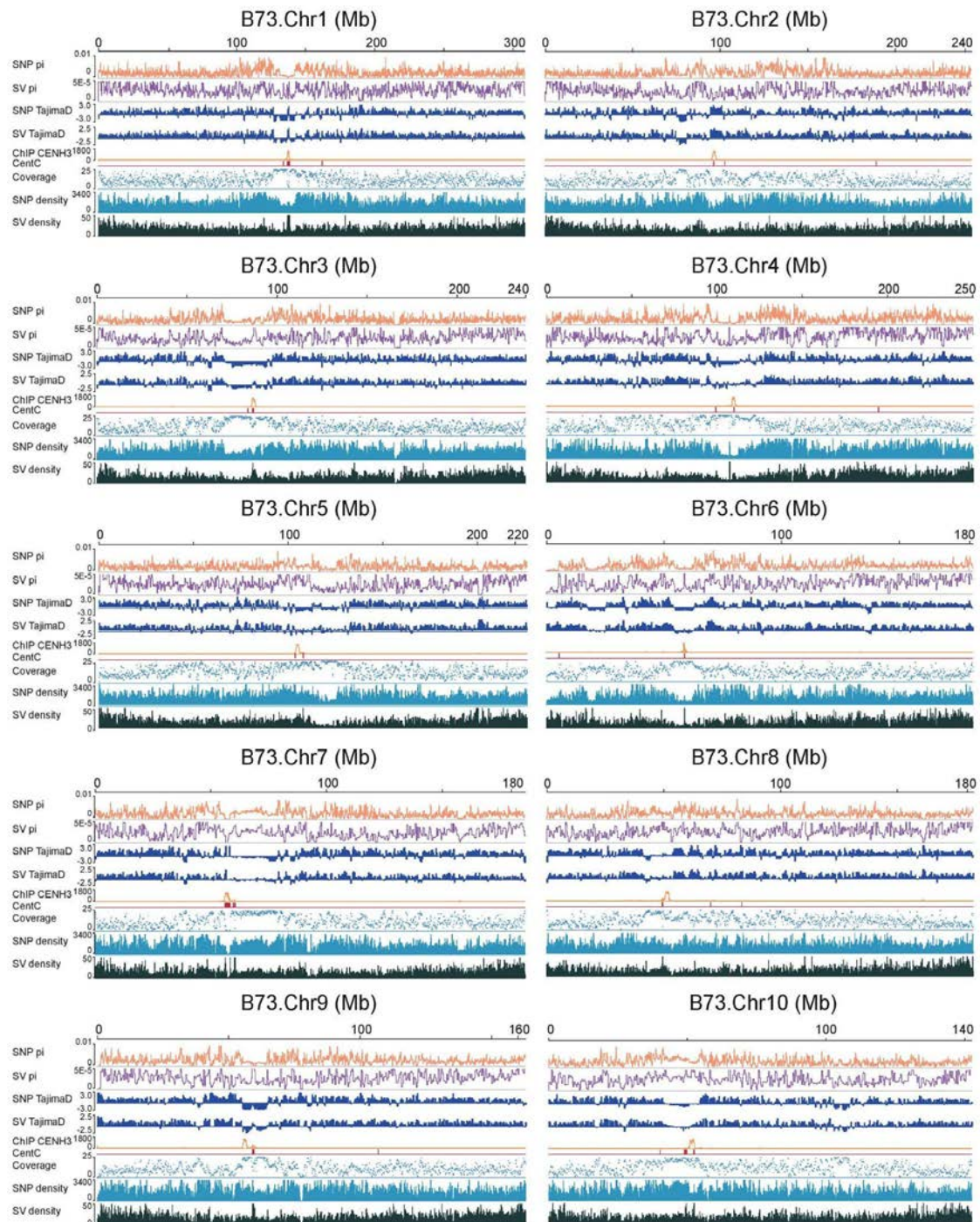
**Supplementary Figure 9: Benchmarking SV discovery and genotyping tools for large-scale population genome assemblies.** Benchmark data were generated using simulated structural variants (SVs) across 20 rice genomes. For details on the evaluation metrics, see Fig. 5c.





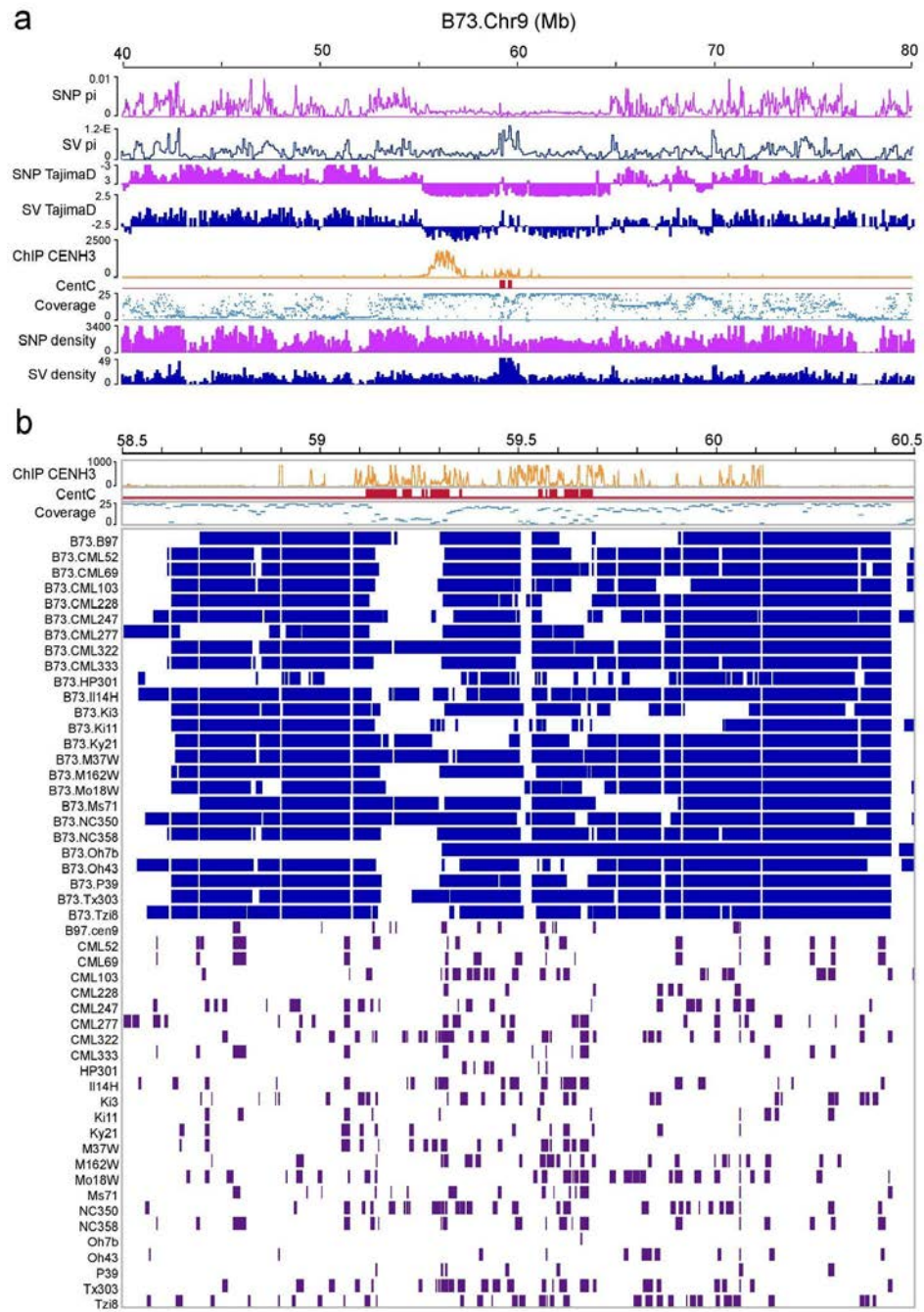


**Supplementary Figure 11: PCR verification of SVGAP-specific SV detection in the maize genome.** The left two gel images show PCR results for 24 randomly selected insertions (relative to B73), and the right image shows results for 16 randomly selected deletions. For each primer pair, the PCR lanes include a DNA ladder, two replicated B73 DNA samples, and two Oh43 DNA samples.

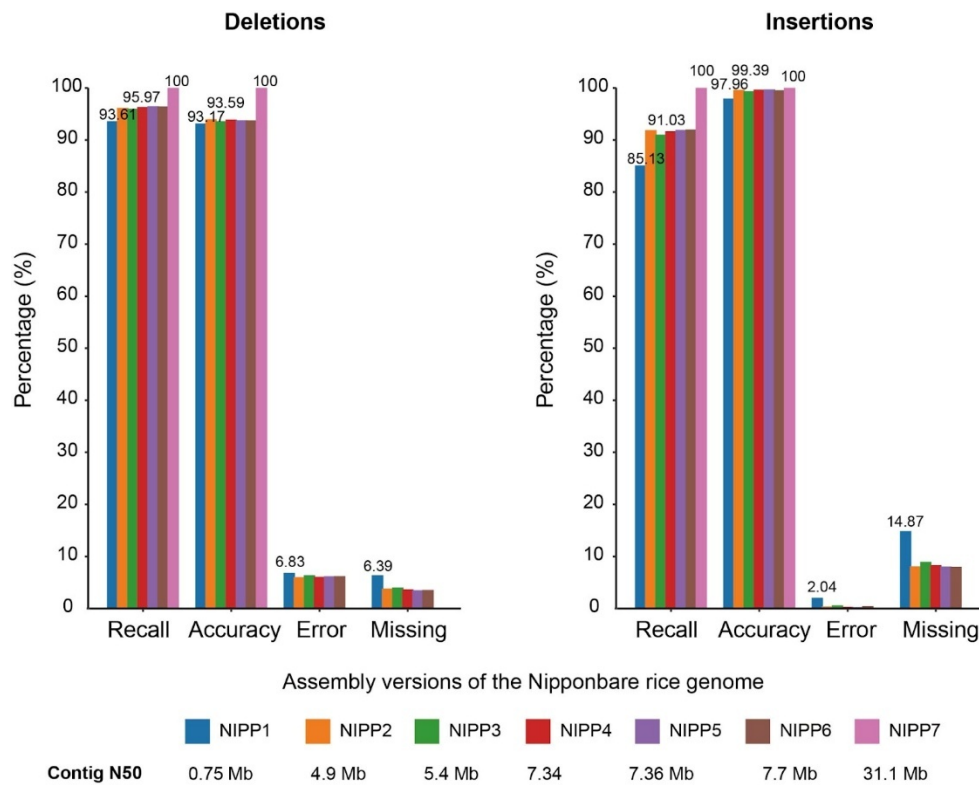


**Supplementary Figure 12: Population properties of genetic variants identified by SVGAP across the maize genome.** The average pairwise diversity ( $\pi$ ), Tajima's D, as well as SNP and SV density were calculated in 100-kb non-overlapping windows along each chromosome. Coverage is represented by the number of genomes that can be syntenically aligned to the B73 reference, calculated for each 10-kb window. Notably, an elevated coverage was observed around the functional centromeric regions across most chromosomes, highlighting the structural and evolutionary conservation of these regions.

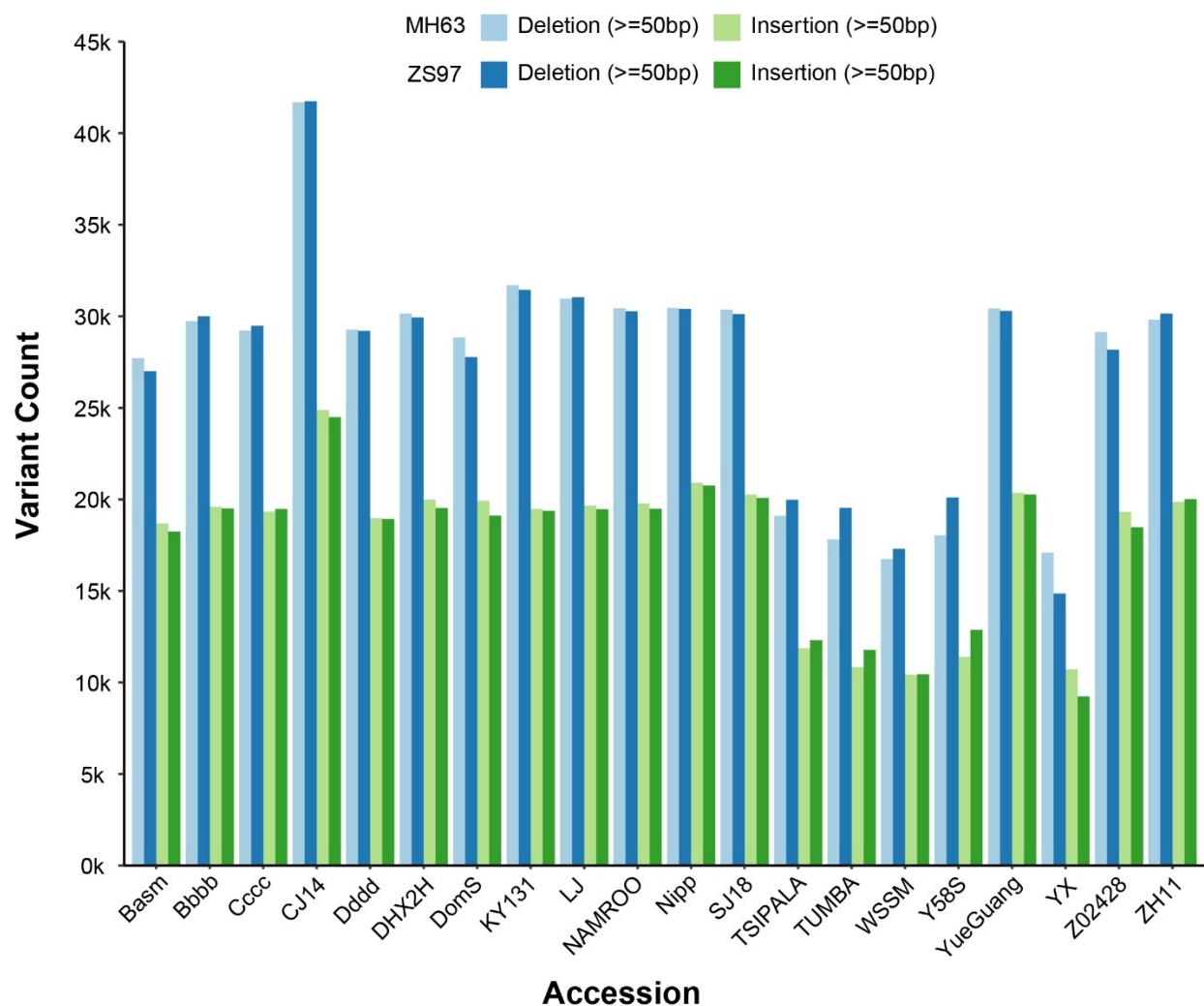




**Supplementary Figure 13: Features of genetic variations and their population properties around the functional centromeric regions of chromosome 9.** **a.** Panels from top to bottom include: (1) average pairwise diversity ( $\pi$ ) for SNPs calculated in non-overlapping 100-bp windows along the chromosome; (2)  $\pi$  calculated for structural variants (SVs); (3) Tajima's D calculated with SNPs; (4) Tajima's D calculated with SVs; (5) CENH3 ChIP-seq peaks; (6) distribution of centromere-specific tandem repeats (CentC); (7) number of genomes aligned to the B73 genome in each 10-kb window; (8) SNP density; and (9) SV density. **b.** Functional centromeric regions are enriched with insertions of centromere-specific retrotransposable elements. Both sequence alignment coverage (blue) and the presence of transposable element (TE) insertions (purple) are shown for 26 maize genomes relative to the B73 reference genome.



**Supplementary Figure 14. Benchmarking SV discovery using Nipponbare rice genome assemblies of varying quality.** Lower assembly quality resulted in decreased recall and accuracy, along with increased error and missing rates.



**Supplementary Figure 15: Impact of reference genome choice on SV discovery across population-scale genome assemblies.** To evaluate how reference genome choice affects SV detection, we compared results from using two rice reference genomes, *MH63* and *ZS97*, across 20 rice accessions. Using *ZS97* as the reference, we identified 91,466 deletions and 86,638 insertions, which is highly comparable (~1% difference) to the 90,978 deletions and 85,162 insertions detected using *MH63* as the reference.

# Selected references

1. Bartenhagen, C. & Dugas, M. RSVSim: an R/Bioconductor package for the simulation of structural variations. *Bioinformatics* **29**, 1679–1681 (2013).
2. Kapitonov, V. V. & Jurka, J. Molecular paleontology of transposable elements in the *Drosophila melanogaster* genome. *Proc Natl Acad Sci USA* **100**, 6569–6574 (2003).
3. Hoskins, R. A. *et al.* The Release 6 reference sequence of the *Drosophila melanogaster* genome. *Genome Res* **25**, 445–458 (2015).
4. Chakraborty, M. *et al.* Hidden genetic variation shapes the structure of functional elements in *Drosophila*. *Nat. Genet.* **50**, 20–25 (2018).
5. Hoyt, S. J. *et al.* From telomere to telomere: The transcriptional and epigenetic state of human repeat elements. *Science* **376**, eabk3112 (2022).
6. Nurk, S. *et al.* The complete sequence of a human genome. *Science* **376**, 44–53 (2022).
7. Song, J.-M. *et al.* Two gap-free reference genomes and a global view of the centromere architecture in rice. *Mol. Plant* **14**, 1757–1767 (2021).
8. Hosmani, P. S. *et al.* An improved de novo assembly and annotation of the tomato reference genome using single-molecule sequencing, Hi-C proximity ligation and optical maps. Preprint at <https://doi.org/10.1101/767764>.
9. Alonge, M. *et al.* Major Impacts of Widespread Structural Variation on Gene Expression and Crop Improvement in Tomato. *Cell* **182**, 145–161.e23 (2020).
10. Hufford, M. B. *et al.* De novo assembly, annotation, and comparative analysis of 26 diverse maize genomes. *Science* **373**, 655–662 (2021).
11. Liao, Y. *et al.* The 3D architecture of the pepper genome and its relationship to function and evolution. *Nat. Commun.* **13**, 3479 (2022).
12. Qin, C. *et al.* Whole-genome sequencing of cultivated and wild peppers provides insights into *Capsicum* domestication and specialization. *Proc. Natl. Acad. Sci. U. S. A.* **111**, 5135–5140 (2014).

13. Kielbasa, S. M., Wan, R., Sato, K., Horton, P. & Frith, M. C. Adaptive seeds tame genomic sequence comparison. *Genome Res.* **21**, 487–493 (2011).
14. Marçais, G. *et al.* MUMmer4: A fast and versatile genome alignment system. *PLoS Comput. Biol.* **14**, e1005944 (2018).
15. Li, H. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics* **34**, 3094–3100 (2018).
16. Song, B. *et al.* AnchorWave: Sensitive alignment of genomes with high sequence diversity, extensive structural polymorphism, and whole-genome duplication. *Proc. Natl. Acad. Sci. U. S. A.* **119**, (2022).
17. Lin, H.-N. & Hsu, W.-L. GSAIalign: an efficient sequence alignment tool for intra-species genomes. *BMC Genomics* **21**, 182 (2020).
18. Nattestad, M. & Schatz, M. C. Assemblytics: a web analytics tool for the detection of variants from an assembly. *Bioinformatics* **32**, 3021–3023 (2016).
19. Goel, M., Sun, H., Jiao, W.-B. & Schneeberger, K. SyRI: finding genomic rearrangements and local sequence differences from whole-genome assemblies. *Genome Biol.* **20**, 277 (2019).
20. Heller, D. & Vingron, M. SVIM-asm: structural variant detection from haploid and diploid genome assemblies. *Bioinformatics* **36**, 5519–5521 (2021).
21. O'Donnell, S. & Fischer, G. MUM&Co: accurate detection of all SV types through whole-genome alignment. *Bioinformatics* **36**, 3242–3243 (2020).