# GSV benchmarking - step by step guide

# Overview

Genomic Sequence Variant (**GSV**) benchmarking evaluates how well different classifier settings distinguish closely related genomes.
This workflow clusters reference genomes from a target species into GSVs, simulates metagenomic samples from those references, and tests multiple Kraken2 and mSWEEP configurations **to maximise recall and precision**.
Off-target genomes from other species in the same family are processed in parallel to measure **false-positive rates**, allowing the pipeline to pinpoint the best combination of:

- Number of GSV clusters
- Kraken2 classifier confidence thresholds
- Count filtration to account for false positive classification

## Table of Contents



# Step 1: On-target genomes

## 1.1 Download all reference genomes

**In progress**

Example script to download genomes from accession list:

```
#!/bin/bash
#SBATCH -t 04:00:00 -n 1 --cpus-per-task=1 --mem=10G -o get_ncbi_genomes.out
module load WebProxy

datasets download genome accession --inputfile accessions_mh_2435.txt --include
genome --filename Mh_genbank.zip
```

## 1.2 Run ANI on all genomes

**In progress**

## 1.3 Use hclust() and cutree() to create GSVs

**In progress**

# Step 2: Off-target genomes

## 1.1 Identify genomes for same family as target species

**In progress**

## 1.2 Remove accessions

**In progress**

Get a list of accessions for all genomes in the same family as your target genome, then use the python script "INSERT_HERE" on that metadata sheet to remove accessions for genomes from your target species.

# Step 3: Bechmarking

What you'll need:

1. ANI annotations with clusters from 2-30 (or more)
2. Target genomes
3. Off-target genomes
4. Themisto database
5. Kraken database

## 3.1 Preparing resources

### 3.1.1 Install VARIANT++ conda environment

We'll need to download the VARIANT++ github directory which will include all the necessary scripts and recipe for the conda environment.

I tend to use miniconda installed locally, but you can also just look for the latest anaconda module and load it directly. Once you have conda available, run this:

```
git clone https://github.com/Microbial-Ecology-Group/VARIANTplusplus.git

cd VARIANTplusplus/

conda env create -f envs/VARIANT++_env.yaml

conda activate VARIANT++_env
# if that doesn't work, use "source" instead of "conda" for the command above.
```

There's one software tool, themisto, that comes in the VARIANT++ repository, but unfortunately we have another step to make it available for our benchmarking analysis. We'll get the absolute path to the themisto executable you just downloaded and add that to your $PATH so that it comes up automatically.

```
# Navigate to directory from the VARIANT++ dir
cd bin/

# Get path
pwd

# Add that directory to your $PATH variable
# Replace "/path/to/your/wd/" with your path
echo 'export PATH="/path/to/your/wd/:$PATH"' >> ~/.bashrc

# Now run the bash.rc file
source ~/.bashrc
```

### 3.1.2 Make themisto database

We'll use your target genomes to create a themisto database.

```
ls -d /scratch/user/u.ed124096/Mh_benchmarking/Mh_2435_genomes/*.fna >
2025_themisto_input_Mh_sequences.txt

themisto build -k 21 -i 2025_themisto_input_Mh_sequences.txt -o
2025_themisto_index_no --temp-dir tmp -t 2 --mem-gigas 4
```

### 3.1.3 Confirm ANI clusters are in same order as themisto db

Refer back to the file we made in R with all genomes in the first column and subsequent colums named "k_#". Open it with excel and make sure you sort all the rows to that they match the order of the genomes in the file you just made during the creation of the themisto database, in this case `2025_themisto_input_Mh_sequences.txt`. Save it with the fixed order, we'll be pointing to this below.

### 3.1.4 Download kraken database

The coreNT kraken database. It's about 240 GB and can be downloaded like this:

```
# Download database
wget https://genome-idx.s3.amazonaws.com/kraken/k2_core_nt_20241228.tar.gz

# Make directory for db contents
mkdir -p k2_core_nt_20241228

# unzip it
tar -xzvf k2_core_nt_20241228.tar.gz -C k2_core_nt_20241228/
```

## 3.2 Benchmarking with on target genomes (recall/precision) and off target genomes (false positive rate)

### 3.2.1 Make scripts

1. Modify the `params.txt` file to point to the corresponding directories to your genomes, the themisto database, ANI annotatinos, and kraken database.
    - Only fill out the variables as present in the template, don't add any quotes, spaces, or comments after the variables.
    - I recommend running a test with only a few simulations, low number of genomes per GSV, and maybe reduce the kraken confidence scores that's included. Once you can get through all steps, you can come back and modify the `params.txt` file again with the full simulation paramters.
2. Run the `generate_simulated_samples_byConf.py` script which will make three seperate scripts that we'll run in portions.

Command to run in terminal (don't need sbatch)

```
python generate_simulated_samples_byConf.py --benchmarking_params params.txt
```

This will output three scripts. They'll have different names depending on what you used for the `output_name_prefix` parameter in the `params.txt` file.

- script_build_reads_test_benchmarking.sh
- script_kraken_extract_split_test_benchmarking.sh
- script_themisto_msweep_test_benchmarking.sh

### 3.2.2 Run script to build simulated reads

The scripts we just made are way too long to run at once, so we'll use the script `split_script_iteration_markers.py`. This will break up the processes evenly across any given number of scripts.

For each of the three scripts, we'll need to modify the `split_script_iteration_markers.py` script and change the parameters for the first variable at the top of the script, `sbatch_header_template`. For the first step, you can use something like this:

```
sbatch_header_template = (
    "#!/bin/bash\n"
    "#SBATCH -J {jobname}\n"
    "#SBATCH -o scripts/log_{jobname}.out\n"
    "#SBATCH -t 24:00:00\n"
    "#SBATCH --mem=40G\n"
    "#SBATCH --nodes=1\n"
    "#SBATCH --ntasks=1\n"
    "#SBATCH --cpus-per-task=48\n"
    "#SBATCH --ntasks-per-node=1\n\n"
)
```

The main parameters of importance is the time `-t` and the memory `--mem` for each script that is created.

In this python command, we'll point to the first script, "script_build_reads_test_benchmarking.sh", then specify the number of scripts to make, and the prefix for the scripts that will be made.

```
python split_script_iteration_markers.py script_build_reads_test_benchmarking.sh
20 buildReads_byConf
```

Read the output text, you might notice if you used too high a number of some scripts have 0 lines or "0 marker lines". This means that there weren't enough code chunks to split into that many scripts. These scripts are now in a new directory, `scripts/`.

If, for example, only 13 of the 20 scripts had commands then we can use the following command to submit those scripts using sbatch. make sure to change the prefix to the left of the word "part" to match the prefix you used above.

```
for i in {1..13}; do
  sbatch "scripts/buildReads_byConf_part_${i}.sbatch"
done
```

You'll now have to wait for those scripts to finish running. Remember to do a good testing run, so you have an idea of how long it will take per simulated. Maybe take chunks from one of the scripts and try running all the commands to see how long you'll need.

Once everything is done running, you can consider erasing the temporary directory specified by the `tmp_build` paramater, in this case "temp_conf". This can end up taking a lot of space so keep an eye on your

storage space. You can do this at the end of each step, just make sure everything completed succesfully by looking at the end of the logs in the `scripts/` directory.

### What if some of my scripts didn't completely finish?

If you look at the logs and see that they were cutoff due to time, you should be able to identify the last sample that was succesfully run by that script and erase everything up to that set of commands. Then you can resubmit, but try to calculate how much time would actually be needed before re-submitting.

### 3.2.3 Run script to run kraken and extract reads

This next section requires alot more resources because we'll be using kraken2 and the core nt database. Loading that large database requires at least 250G of memory. We could add "--memory-mapping" as a kraken2 flag to reduce the required memory, but it ends up taking way longer, so better to just request the full amount.

We'll modify the `split_script_iteration_markers.py` script again, to update the sbatch header so that it includes the 250G in memory. Remember to change the time based on your testing.

```
sbatch_header_template = (
    "#!/bin/bash\n"
    "#SBATCH -J {jobname}\n"
    "#SBATCH -o scripts/log_{jobname}.out\n"
    "#SBATCH -t 24:00:00\n"
    "#SBATCH --mem=250G\n"
    "#SBATCH --nodes=1\n"
    "#SBATCH --ntasks=1\n"
    "#SBATCH --cpus-per-task=48\n"
    "#SBATCH --ntasks-per-node=1\n\n"
)
```

Then, we'll run the script to split up the next series of commands from the second script, `script_kraken_extract_split_test_benchmarking.sh`.

```
python split_script_iteration_markers.py
script_kraken_extract_split_test_benchmarking.sh 20 kraken_byConf
```

Submit them as before, remembering to update the prefix.

```
for i in {1..20}; do
  sbatch "scripts/kraken_byConf_part_${i}.sbatch"
done
```

### 3.2.4 Run script to classify GSVs in simulated samples

This final step will be similar, but requires much less memory. So update the `split_script_iteration_markers.py` again to look something like this:

```
sbatch_header_template = (
    "#!/bin/bash\n"
    "#SBATCH -J {jobname}\n"
    "#SBATCH -o scripts/log_{jobname}.out\n"
    "#SBATCH -t 24:00:00\n"
    "#SBATCH --mem=10G\n"
    "#SBATCH --nodes=1\n"
    "#SBATCH --ntasks=1\n"
    "#SBATCH --cpus-per-task=48\n"
    "#SBATCH --ntasks-per-node=1\n\n"
)
```

Then, we'll run the script to split up the next series of commands from the third script, `script_themisto_msweep_test_benchmarking.sh`.

```
python split_script_iteration_markers.py
script_themisto_msweep_test_benchmarking.sh 20 classify_byConf
```

Submit them as before, remembering to update the prefix.

```
for i in {1..20}; do
  sbatch "scripts/classify_byConf_part_${i}.sbatch"
done
```

# Step 4: Analyzing benchmarking results

**In progress**