Project update for our replication of "Soil microbiome indicators can predict crop growth response to large-scale inoculation with arbuscular mycorrhizal fungi."

1. Project Overview

We are working to replicate the analysis pipeline described in Lutz et al. (2023), "Soil microbiome indicators can predict crop growth response to large-scale inoculation with arbuscular mycorrhizal fungi," published in Nature Microbiology.

> Lutz, S., Bodenhausen, N., Hess, J. *et al.* Soil microbiome indicators can predict crop growth response to large-scale inoculation with arbuscular mycorrhizal fungi. *Nat Microbiol* 8, 2277–2289 (2023). https://doi.org/10.1038/s41564-023-01520-w
>
> Original project GitHub (linked in paper): http://www.ebi.ac.uk/ena
> Data source (linked in paper):
> > Soil microbiome samples:
> > https://www.ebi.ac.uk/ena/data/view/PRJEB53587
> > Root microbiome samples:
> > https://www.ebi.ac.uk/ena/data/view/PRJEB56590

Summary and key findings of the paper:

The proof-of-concept study conducted by Lutz et al. (2023) shows that soil microbiome indicators can reliably predict the effectiveness of arbuscular mycorrhizal fungi (AMF) inoculation in enhancing maize (*Zea mays* L.) growth under field conditions. In a large-scale, multi-farm field study encompassing 54 maize fields in Northern Switzerland, inoculation with *Rhizoglomus irregulare* (strain SAF22) resulted in highly variable mycorrhizal growth responses (MGR), ranging from -12% to +40%. Notably, only in 14 fields (~25%) were significant positive growth responses for maize observed.

Interestingly, the investigators found that soil fungal operational taxonomic units (sOTUs) outperformed conventional soil parameters in accurately predicting MGR outcomes. They initially screened 52 soil variables and hundreds of fungal sOTUs from long-read sequencing of the ITS (Internal Transcribed Spacer) region. Using random forest, stepwise model selection (AIC), and glmulti exhaustive model comparison, they narrowed the soil variables and the fungal sOTUs to 15 and 13, respectively. When these 28 predictors were integrated into a multiple linear regression model, the model accurately explained up to 86% of the variation in MGR, with fungal sOTUs explaining the majority of the variance, i.e., 53%, compared to soil parameters contributing 29%. Year and other unknown factors accounted for the remaining 4% of the variation in MGR.

The primary mechanism for maize plant growth and yield improvement was not enhanced nutrient acquisition but suppression of soil-borne fungal pathogens. AMF inoculation enhances maize plant growth primarily by suppressing soil pathogens, particularly in degraded soils with low microbial biomass carbon (MBC). Fields that exhibited strong maize growth responses to AMF inoculation had elevated levels of pathogenic fungi such as *Trichosporon* and *Olpidium*, which were significantly reduced post-inoculation. Surprisingly, AMF root colonization or the success rate of the inoculated AMF strain did not correlate with maize plant growth. Hence, soil fungal community composition and function are much stronger predictors of AMF inoculation success than traditional soil fertility metrics like phosphorus or nitrogen.

This research lays the foundation for diagnostic tools to inform large-scale AMF field inoculation decisions based on rapid soil microbiome profiling. Soil fungal community profiling can predict AMF inoculation success under field conditions, offering a practical tool for microbiome-based agricultural management. Such predictive microbiome-based biotechnological approaches can potentially increase the agronomic yield and farmers' profitability while contributing to more sustainable and resilient agricultural crop production systems.

The method they used involves:

- Processing PacBio sequencing data for both soil and root microbiome samples
- Using DADA2 (Divisive Amplicon Denoising Algorithm) to process the sequencing data and identify high-quality amplicon sequence variants (ASVs)
- Clustering these ASVs into operational taxonomic units (OTUs)
- Relating microbiome composition, such as fungal pathogens, to plant growth response

The goal of our project is to replicate their bioinformatics pipeline. This includes the DADA2 processing step, which is very computationally intensive and takes a long time to process.

2. Exploratory analysis of the data

The "raw" sequencing data (linked in the paper and provided by the European Nucleotide Archive) come from two sample types:

1. Soil microbiome samples: Sequenced using general ITS1F/ITS4 primers targeting the entire ITS region
2. Root microbiome samples: Sequenced using AMF-specific primers (SSUmCf/LSUmBr) that target a part of the small ribosomal subunit, the entire ITS region, and part of the large ribosomal subunit

- The data that is provided includes four soil sample files and three root sample files (as described by the paper)
- The data in the EBI are all labeled "soil microbiome," so we went to the paper to decipher which were from the soil and which were from the root. As the paper states:
  "The raw sequencing data are stored in the European Nucleotide Archive (http://www.ebi.ac.uk/ena) under accession numbers PRJEB53587 (soil microbiome) and PRJEB56590 (root microbiome)."
- So, checking PRJEB53587 (from the paper, this is from the soil), we find:
  - ERR10096273, which is associated with the files named microbials9
  - ERR10200993, which is associated with the files named microbials11
  - ERR10096271, which is associated with the files named microbials1
  - ERR10096272, which is associated with the files named microbials8
- Checking PRJEB56590 (which we found is the root from the paper), we find:
  - ERR10322985, which is associated with the files named microbials10
  - ERR10211983, which is associated with the files named microbials5
  - ERR10322984, which is associated with the files named microbials7

For this project, we are focusing on the root samples.

Initial examination of the sequence files:

- Fastq File sizes range from around 190MB to 4GB, and Bam file sizes range from 40MB to nearly 1GB

```
-rw-r-----. 1 sadiya student 1154M Apr 24 12:00 ERR10096271.fastq
-rw-r-----. 1 sadiya student  451M Apr 24 12:01 ERR10096272.fastq
-rw-r-----. 1 sadiya student  190M Apr 24 12:01 ERR10096273.fastq
-rw-r-----. 1 sadiya student 3884M Apr 24 12:01 ERR10200993.fastq
-rw-r-----. 1 sadiya student  590M Apr 24 12:03 ERR10322983.fastq
-rw-r-----. 1 sadiya student  816M Apr 24 12:03 ERR10322984.fastq
-rw-r-----. 1 sadiya student 1065M Apr 24 12:03 ERR10322985.fastq
-rw-r-----. 1 sadiya student  285M Apr 24 12:05 microbials10.ccs.bam
-rw-r-----. 1 sadiya student  914M Apr 24 12:02 microbials11.ccs.bam
-rw-r-----. 1 sadiya student  266M Apr 24 12:02 microbials1.ccs.bam
-rw-r-----. 1 sadiya student  150M Apr 24 12:04 microbials5.ccs.bam
-rw-r-----. 1 sadiya student  187M Apr 24 12:04 microbials7.ccs.bam
-rw-r-----. 1 sadiya student  100M Apr 24 12:02 microbials8.ccs.bam
-rw-r-----. 1 sadiya student   40M Apr 24 12:02 microbials9.ccs.bam
```

- Read counts range from about 128,000 to around 2.9 million reads per file

```
12:19:32 sadiya$ for i in *.fastq; do echo $(cat $i | wc -l)/4 | bc;
echo; done
ERR10096271.fastq
733485
ERR10096272.fastq
287530
ERR10096273.fastq
127980
ERR10200993.fastq
2886255
ERR10322983.fastq
203336
ERR10322984.fastq
266311
ERR10322985.fastq
```

```
        346428
```

- The data are PacBio CCS (Circular Consensus Sequencing) reads. These are long reads (approximately 500-1800bp). We checked the files with FastQC and verified that they are high quality.

3. Pre-processing was previously run on the data we were provided

- The files are already demultiplexed, as we discovered by:
    - The read headers contain unique identifiers like @ERR10096271.1 m54073_181025_124715/4391112/ccs
    - There are individual files for each sample type
- The sequencing data is in fastq format, containing both sequence data and quality scores. According to the phred scores in the data (and fastqc analysis), the quality of the reads is high.

Since it was already completed before the data they gave us (and they don't provide data from before this step), we skipped the demultiplexing step (01_Demultiplex) on GitHub and started with the DADA2 processing step.

4. GitHub Repository

The provided GitHub repository (https://github.com/PMI-Basel/Lutz_et_al_Predicting_crop_yield) contains:

- Demultiplexing scripts and configuration files (01_Demultiplex/)
- DADA2 pipeline script (02_dadapipe/ASV_PacBio.R and run_ASV_cluster.sh)
- Output tables from OTU clustering (03_ASV_tables/)
- MGR prediction (04_MGR_prediction/)
- Quality control reports (03_ASV_tables/quality_check/)

The repository (and paper) contains the basic scripts needed to replicate the analysis, but not detailed documentation about how to execute them or specific parameters for different sample types.

Also, the primer files (primer.fasta) in GitHub contain only the primers from the root sample types. Luckily, the correct primers were listed in the paper, so for the soil samples, we made a FASTA containing those (we are focusing on the root samples, but just in case).

***Of note, in the paper, they state that for the soil samples, they used primers ITS1F (5'-CTTGGTCATTTAGAGGAAGTAA-3') and ITS4 (5'-TCCTCCGCTTATTGATATGC-3'). This is mentioned in the Methods section under "Soil microbiome sequencing." They used (5'-TATYGYTCTTNAACGAGGAATC-3') and (5'-AACACTCGCAYAYATGYTAGA-3 for the root samples. We wrote a script to verify whether these are the data's primers. Here is the output:

```
File: .//ERR10096271.fastq
  Total reads: 733485
```

```
    Primer: AACACTCGCAYAYATGYTAGA → 0 reads (0.00%)
    Primer: TCCTCCGCTTATTGATATGC → 370720 reads (50.54%)
    Primer: CTTGGTCATTTAGAGGAAGTAA → 348210 reads (47.47%)
    Primer: TATYGYTCTTNAACGAGGAATC → 0 reads (0.00%)
  Total with any primer: 670511 (91.41%)
File: .//ERR10096272.fastq
  Total reads: 287530
    Primer: AACACTCGCAYAYATGYTAGA → 0 reads (0.00%)
    Primer: TCCTCCGCTTATTGATATGC → 135629 reads (47.17%)
    Primer: CTTGGTCATTTAGAGGAAGTAA → 131712 reads (45.81%)
    Primer: TATYGYTCTTNAACGAGGAATC → 0 reads (0.00%)
  Total with any primer: 255853 (88.98%)
File: .//ERR10096273.fastq
  Total reads: 127980
    Primer: AACACTCGCAYAYATGYTAGA → 0 reads (0.00%)
    Primer: TCCTCCGCTTATTGATATGC → 59890 reads (46.80%)
    Primer: CTTGGTCATTTAGAGGAAGTAA → 56356 reads (44.04%)
    Primer: TATYGYTCTTNAACGAGGAATC → 0 reads (0.00%)
  Total with any primer: 113374 (88.59%)
File: .//ERR10200993.fastq
  Total reads: 2886255
    Primer: AACACTCGCAYAYATGYTAGA → 0 reads (0.00%)
    Primer: TCCTCCGCTTATTGATATGC → 1283059 reads (44.45%)
    Primer: CTTGGTCATTTAGAGGAAGTAA → 1244961 reads (43.13%)
    Primer: TATYGYTCTTNAACGAGGAATC → 0 reads (0.00%)
  Total with any primer: 2525978 (87.52%)
File: .//ERR10322983.fastq
  Total reads: 203336
    Primer: AACACTCGCAYAYATGYTAGA → 89561 reads (44.05%)
    Primer: TCCTCCGCTTATTGATATGC → 95521 reads (46.98%)
    Primer: CTTGGTCATTTAGAGGAAGTAA → 84706 reads (41.66%)
    Primer: TATYGYTCTTNAACGAGGAATC → 73708 reads (36.25%)
  Total with any primer: 187812 (92.37%)
File: .//ERR10322984.fastq
  Total reads: 266311
    Primer: AACACTCGCAYAYATGYTAGA → 116698 reads (43.82%)
    Primer: TCCTCCGCTTATTGATATGC → 126538 reads (47.52%)
    Primer: CTTGGTCATTTAGAGGAAGTAA → 126156 reads (47.37%)
    Primer: TATYGYTCTTNAACGAGGAATC → 108740 reads (40.83%)
  Total with any primer: 260467 (97.81%)
File: .//ERR10322985.fastq
  Total reads: 346428
    Primer: AACACTCGCAYAYATGYTAGA → 150106 reads (43.33%)
    Primer: TCCTCCGCTTATTGATATGC → 167654 reads (48.40%)
    Primer: CTTGGTCATTTAGAGGAAGTAA → 160561 reads (46.35%)
    Primer: TATYGYTCTTNAACGAGGAATC → 135867 reads (39.22%)
  Total with any primer: 337818 (97.51%)
```

So, interestingly (and this is not in line with what was stated in the paper), the soil samples contained only soil-specific primers, with around 90% of the reads having at least one. There are no reads in the soil that have the AMF primers.
*** However, in the root samples, all four primers are found in large numbers, and more than 92% have at least one primer.

To handle this, we put all four primers in the primer.fasta files for each of the soil samples (again, we are focusing on the root samples, but if they become useful, we are concurrently making changes for the soil samples as well).

Computational/Time Challenges

The first DADA2 processing attempt took over 72 hours, even though we ran it on a high-performance system with 48 cores and 256GB of RAM (we didn't change anything initially from the script). Due to our time and computational constraints, we checked some sources and made the following changes, which should reduce the computational time without significantly affecting the results.

- Reduced BAND_SIZE parameter from 32 to 16 in error learning and sequence denoising steps. Error learning involves training models to identify and correct errors in sequences, while denoising focuses on removing noise or unwanted artifacts from sequences. According to documentation, this should significantly speed up the processing without too much cost on accuracy. If we were doing this study ourselves, we should keep the BAND_SIZE at 32, but for this project (and the time limits), 16 seems more appropriate.
- Set MAX_CONSIST=10 to prevent a high number of time-demanding iterations – the original version iterated many times, making completing the project seem less likely.
- We used only 100 million bases (nbases=1e8) for error rate estimation, which aligns with what others reported elsewhere (but not this paper).
- We initially used all available cores(48), but it seemed to "hang" at various times in processing (checking with htop), so we reduced the CPU cores used for dada2 to 75% of the total, and the issue went away.
- We added timing measurements for each major processing step
- We implemented more detailed status messages throughout the pipeline
- We implemented checkpoint saving to allow us to restart later if it crashed.

1. Separate processing for root and soil samples:
   While our part of the project is just focusing on the root samples, we ran the soil on a separate computer simultaneously, as it didn't take any extra time to do so, just in case it is helpful to us later in the project.
   - Root samples (microbials5, microbials7, microbials10) processed together
   - Soil samples (microbials1, microbials8, microbials9, microbials11) processed together
2. The reason to process the root and soil separate:
   - Downstream analyses use these datasets separately
3. Project organization:
   - We keep the same GitHub directory structure
   - We run each sample type on a separate machine to parallelize the process
4. Modularization/parallel limitations:
   - Each sample type (soil or root) has to be processed as a group
   - The different steps of the pipeline use the output from the previous step, so it isn't easy to parallelize further. Also, according to all the documentation we found, only one of the steps seems very time-intensive (dada2). The later parts combine the results from the dada2 output and do further processing

from there. Some sections may be able to be split after that, but we first need the output from the first step, which looks like it is the most time-consuming anyway, so we probably wouldn't benefit from it.

We are focusing on the root part. In addition to the primer issue, it seems likely that the GitHub code (as it only contains the soil samples primers) focuses on the soil samples. Some of the filtering and processing parameters in the scripts are different from what they used for the root part. These are good examples of a) why reproducibility can be difficult and b) why it is essential to record every detail in processing when working on a research project.

Also, it is essential to keep track of all versions of all tools or make a Docker container available to reproduce the work. Some tool versions have changed, and recreating the environment to reproduce the work was challenging.

For example:

The tool used by the script DECIPHER has changed its API.

```
      "Error: 'IdClusters' is not an exported object from 'namespace: DECIPHER'"
```

Checking the documentation, the IdClusters method is now called Clusterize. So, we checked the documentation and updated it accordingly:

```
ASV97_cluster <- DECIPHER::Clusterize(ASV_dist, cutoff = 0.03, processors = nproc)
ASV98_cluster <- DECIPHER::Clusterize(ASV_dist, cutoff = 0.02, processors = nproc)
ASV99_cluster <- DECIPHER::Clusterize(ASV_dist, cutoff = 0.01, processors = nproc)
```

Somehow, however, the Clusterize command is not working correctly. So, we decided to try a different version that has IdClusterize:

```
> if (!requireNamespace("BiocManager", quietly = TRUE)) + install.packages("BiocManager") >
BiocManager::install("DECIPHER") Bioconductor version 3.20 (BiocManager 1.30.25), R 4.4.3
(2025-02-28) Old packages: 'cli', 'igraph', 'RCurl', 'rhdf5', 'rhdf5filters', 'scales'
Update all/some/none? [a/s/n]:
```

Well, the functions in the packages are updated and changes over years.

We were concerned that "update all" might break the environment. (It was challenging to get set up correctly due to dependency issues and specific versions no longer readily available.). So, I checked and found alternates that should generate the same result. The base R "hclust" and "cutree" functions are already in our R install, and they should be identical or nearly the same:

```
# OTUs using base R clustering print("Clustering into OTUs...")
otu_start_time <- Sys.time() hc <- hclust(as.dist(ASV_dist), method="complete")
ASV97_cluster <- data.frame(cluster = cutree(hc, h=0.03))
ASV98_cluster <- data.frame(cluster = cutree(hc, h=0.02))
ASV99_cluster <- data.frame(cluster = cutree(hc, h=0.01))
```

```
otu_end_time <- Sys.time() print(paste("OTU clustering completed in:",
difftime(otu_end_time, otu_start_time, units="mins"), "minutes"))
```

Also, there are a few errors in the code (in addition to the above). For example, the original code has the following:

```
  histo_unfiltered <- hist(lens, 100, main = paste(run, "unfiltered"))
  assign(paste(run, "histo_unfiltered", sep="_"), histo_unfiltered)

  #names filter
  filts <- file.path(path_run, "filtered", basename(fns))
  assign(paste(run, "_filts", sep=""), filts)

  #filter
  #max Len prevents double and triple CCS reads from passing the filter
  track <- filterAndTrim(nop, filts, minQ=3, minLen=500, maxLen=1800, maxN=0,
rm.phix=T, maxEE=2, multithread = T)

  assign(paste(run, "track", sep="_"), track)
  saveRDS(track, paste(path.rds, run, "_out.RDS", sep=""))

  #sequence length after filtering
  histo_filtered <- hist(lens, 100, main = paste(run, "filtered"))
  assign(paste(run, "histo_filtered", sep="_"), histo_filtered)
```

These two commands create the same histogram. (they are use the same database – “lense”)

So, the plots use the same 'lens' variable to make the plots, showing them as the same (both from the unfiltered data).

Some other things:

The original code has:

```
        dd <- vector()
        for(i in runs){
        dd <- c(dd, get(paste(i, "_dd", sep="")))
        }
```

It was necessary to change vector to list to make it run.

But 'c()' doesn't work. Instead, it needs to be a list:

```
        dd <- list()
        for(i in runs){
        dd[[i]] <- get(paste(i, "_dd", sep=""))
```