**Apscale manual**

Last updated: 08.08.2025, apscale version 4.1.1

Author: Dominik Buchner

**Table of contents**

[**1.** **Introduction** 3](#_Toc205543727)

[**2.** **Installation** 3](#_Toc205543728)

[**3.** **Usage** 4](#_Toc205543729)

[**3.1.** **Scalability and resource requirements** 4](#_Toc205543730)

[**3.2.** **Creating a project** 5](#_Toc205543731)

[**3.3.** **Configuring the settings** 5](#_Toc205543732)

[**3.4.** **Adding data** 7](#_Toc205543733)

[**3.5.** **Running a module – Linear workflow** 7](#_Toc205543734)

[**3.5.1.** **Running the full pipeline** 7](#_Toc205543735)

[**3.5.2.** **Paired-end merging** 8](#_Toc205543736)

[**3.5.3.** **Primer trimming** 8](#_Toc205543737)

[**3.5.4.** **Quality filtering** 8](#_Toc205543738)

[**3.5.5.** **Dereplication** 10](#_Toc205543739)

[**3.6.** **Running a module – Modular workflow** 10](#_Toc205543740)

[**3.6.1.** **Denoising** 11](#_Toc205543741)

[**3.6.2.** **Swarm clustering** 12](#_Toc205543742)

[**3.6.3.** **Replicate merging** 12](#_Toc205543743)

[**3.6.4.** **Negative control filtering** 13](#_Toc205543744)

[**3.6.5.** **Read table generation and sequence grouping** 14](#_Toc205543745)

[**3.7.** **Working with the read data store** 14](#_Toc205543746)

[**3.7.1.** **Apscale analyze** 14](#_Toc205543747)

[**3.7.2.** **Adding sample metadata** 15](#_Toc205543748)

[**3.7.3.** **Adding sequence metadata** 17](#_Toc205543749)

[**3.7.4.** **Correcting species names via GBIF** 18](#_Toc205543750)

[**3.7.5.** **Validating species occurrence via GBIF** 20](#_Toc205543751)

[**3.7.6.** **Uploading datasets to ENA** 22](#_Toc205543752)

[**3.7.7.** **Exporting read tables** 23](#_Toc205543753)

# **Introduction**

Apscale is a metabarcoding pipeline that handles the most common tasks in metabarcoding pipelines like paired-end merging, primer trimming, quality filtering, denoising, swarm and threshold-based clustering as well as basic data handling operations such as replicate merging and the removal of reads found in the negative controls. It uses a simple command line interface and is configured via a single configuration file. To add metadata to the dataset, a simple, browser-based interface has been introduced in version 4.0. Apscale automatically uses the available resources on the machine it runs on while still providing the option to use less if desired. All modules can be run on their own or as a comprehensive workflow.

Several different programs are called within the workflow. These include vsearch (Rognes et al. 2016), cutadapt (Martin 2011) and swarm (Mahé et al. 2021). Please cite those accordingly, when using apscale. A DuckDB backend was introduced in version 4.1. Please also cite the authors of DuckDB, they build an amazing tool that makes the current developments in Apscale possible.

Apscale (Buchner et al. 2022) has also been published and we are happy if we are cited too.

# **Installation**

Apscale can be installed on all common operating systems (Windows, Linux, MacOS). Apscale requires Python 3.11 or higher and can be easily installed via pip in any command line:

pip install apscale

To update apscale run:

pip install --upgrade apscale

Apscale calls vsearch as well as swarm for multiple modules. Both programs should be installed and be in PATH to be executed from anywhere on the system.

Check the vsearch and swarm Github pages for further info:

<https://github.com/torognes/>vsearch

<https://github.com/torognes/swarm>

To check if every is correctly set up, please type this into your command line:

vsearch --version

swarm --version

It should return messages similar to these:

Ein Bild, das Text, Screenshot, Schrift, Schwarz enthält.

KI-generierte Inhalte können fehlerhaft sein.

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Further dependencies – cutadapt:

Apscale also calls cutadapt with the primer trimming module. Cutadapt should be downloaded and installed automatically with the Apscale installation. To check this, type:

cutadapt --version

and it should return the version number, for example:



# **Usage**

## **Scalability and resource requirements**

The main strength of the Apscale pipeline lies in its ability to analyze large to massive datasets with modest resource requirements. In particular, the later modules—such as the generation of read tables—can effortlessly handle millions of distinct sequences across thousands of samples. While Apscale runs on virtually any system, we recommend a minimum of 16 GB of RAM for processing large datasets. Apscale is parallelized by default, automatically utilizing all available CPU cores. Additionally, when memory is insufficient, Apscale will spill data to disk. Although this may slow down processing, it ensures that even the most memory-intensive tasks can be completed.

## **Creating a project**

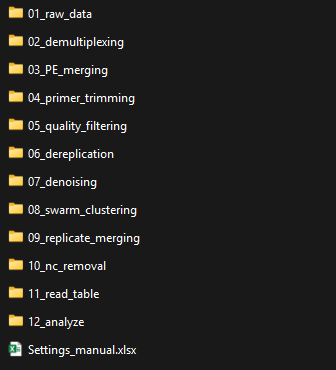
Apscale is organized into projects. A project is a directory that contains subdirectories for the different steps of the pipeline, along with a settings file. All data generated by Apscale will be saved in the corresponding subdirectory within the project’s file system.

To create a new project, run:

apscale --create\_project PROJECT\_NAME

This command will generate a folder structure similar to the one shown below. If only a project name is provided, Apscale will create the project in the current working directory. To specify a different location, include the full path in the project name:





## **Configuring the settings**

All settings required by Apscale are configured within the project’s settings file. Each processing step has its own sheet in the document. The first tab you’ll see when opening the settings file is "0\_general\_settings". In this tab, you can define how many CPU cores Apscale should use for processing. By default, it uses the total number of available cores minus two.

You can also set the gzip compression level here. Apscale compresses all output files to conserve disk space. The default compression level is 6, which is suitable for most use cases. If you need to save more space, you can increase it to 9—this will produce smaller files but may slow down processing.

Most settings come with default values, except for those in the “04\_primer\_trimming” and “05\_quality\_filtering” modules. For details on the required input, please refer to the corresponding sections.

|  |  |  |  |
| --- | --- | --- | --- |
| Setting | Default | Possible values | Effect |
| cores to use | Available cores – 2 | 1 – available cores | Manages parallelization in Apscale |
| compression level | 6 | 1 – 9 | Controls the gzip compression level. Lower values produce larger files but allow for faster processing; higher values result in smaller files at the cost of increased processing time. |

## **Adding data**

Apscale processes demultiplexed paired-end sequencing data. It requires one pair of files per sample, with any consistent naming pattern (e.g., sample1\_f1.fastq.gz and sample1\_r1.fastq.gz). Input files do not need to be unzipped—Apscale can handle gzip-compressed files directly.

If your data is already demultiplexed, place the read files in the 02\_demultiplexing/data directory. If additional demultiplexing is required (e.g., when using inline barcodes), we recommend storing the raw data in 01\_raw\_data/data and writing the output of your demultiplexing script to 02\_demultiplexing/data. This ensures that all data remains organized within the project’s directory structure.

Note that Apscale does not perform demultiplexing itself, as there are many different tagging and barcoding schemes. However, we provide a dedicated tool for this purpose called demultiplexer2, available at: <https://github.com/DominikBuchner/demultiplexer2>.

## **Running a module – Linear workflow**

The steps in the following section will be executed for all projects in a linear sequence. The later modules are modular and can be run or skipped in any combination, depending on the user's objectives. If you are already in the project directory, all commands follow the same logic. Specifying the project path is optional and only necessary if you are running the command from outside the project directory.

apscale --COMMAND [project\_path]

Each processing step also generates a logfile that records detailed information about the step’s execution..

## **Running the full pipeline**

If the settings are fully configured and you want to run the complete analysis, all pipeline steps can be executed with a single command. This will run the entire pipeline using the defined settings. For more control, individual steps can also be run independently.

apscale --run\_apscale [project\_path]

## **Paired-end merging**

The first step performed by Apscale is merging paired-end reads using vsearch. The default settings are fairly relaxed to merge the largest possible portion of reads, as quality filtering is handled in later steps.

To run this module, use the following command:

apscale --pe\_merging [project\_path]

|  |  |  |  |
| --- | --- | --- | --- |
| Setting | Default | Possible values | Effect |
| maxdiffpct | 25 | 0-100 | Maximum percentage of non-matching nucleotides allowed in the overlap region |
| maxdiffs | 199 |  | Maximum number of non-matching nucleotides allowed in the overlap reagion |
| minovlen | 5 |  | Minimum overlap between the merged reads |

## **Primer trimming**

The next step performed by Apscale is primer trimming, which removes the primers used for target amplification since they do not contain biologically relevant information.

To run this module, use the following command:

apscale --primer\_trimming [project\_path]

|  |  |  |  |
| --- | --- | --- | --- |
| Setting | Default | Possible values | Effect |
| P5 Primer (5' - 3') | - | - | Primer that can be found at the P5 end of the reads. Usually, the forward primer |
| P7 Primer (5' - 3') | - | - | Primer that can be found at the P7 end of the reads. Usually, the forward primer |
| anchoring | False | True / False | Whether the read starts directly with the primer sequence or not. If set to False, Apscale will, for example, locate the primer sequence even if it appears after an inline tag. |

## **Quality filtering**

After primer trimming, Apscale performs quality filtering. This step filters out reads with an expected error higher than the threshold defined in the settings, as well as sequences whose lengths fall outside the specified target range. Typically, we use a tolerance of ±10 bases around the target length to allow for some biological variation while removing artifacts such as primer dimers.

To run this module, use the following command:

apscale --quality\_filtering [project\_path]

|  |  |  |  |
| --- | --- | --- | --- |
| Setting | Default | Possible values | Effect |
| maxEE | 1 | 0 - infinity | Maximum number of expected errors allowed per read before it is filtered out. For example, a value of 1 means that all reads with an expected error greater than or equal to 1 will be removed. |
| min length | - | - | Minimum length of the sequence |
| max length | - | - | Maximum length of the sequence |

## **Dereplication**

Before running the modular workflow, the reads from all samples must be dereplicated. This step does not alter the data itself but optimizes how it is stored. The output is a FASTA file containing all unique sequences with size annotations (e.g., >seq\_1;size=100).

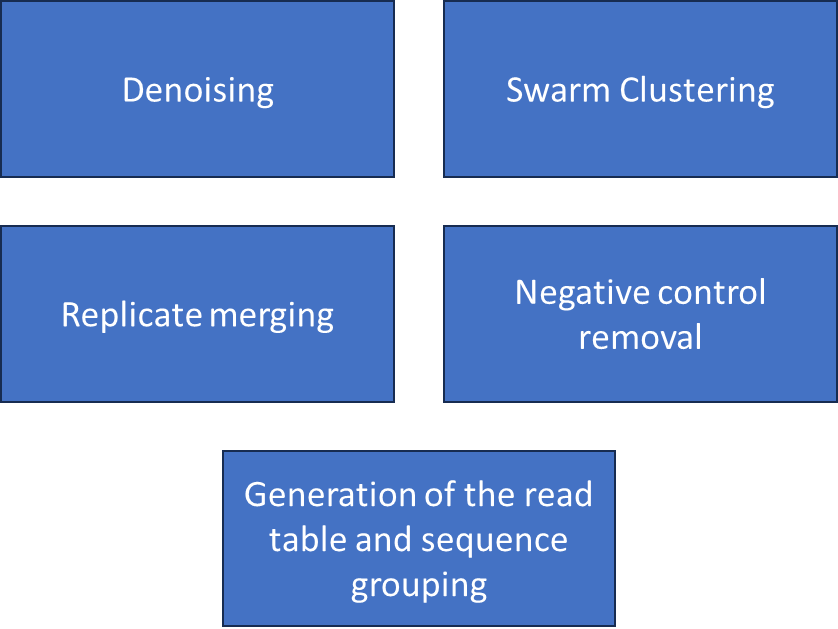
To run this module, use the following command:

apscale --dereplication [project\_path]

|  |  |  |  |
| --- | --- | --- | --- |
| Setting | Default | Possible values | Effect |
| minimum sequence abundance | 1 | 1 - infinity | Minimum abundance required for a sequence to be retained in the dataset. For example, a value of 2 will remove all singletons (sequences that appear only once) |

## **Running a module – Modular workflow**

After dereplication, Apscale has been modularized starting with version 4.0.0, meaning that all subsequent modules are optional. These steps can be freely combined, skipped, or disabled entirely, depending on your specific analysis goals. Each module uses the output of the previously executed step as its input, ensuring a flexible and customizable workflow.



## **Denoising**

The denoising module performs sequence denoising using vsearch, processing each sample file individually. Pooling is intentionally avoided to ensure that the resulting sequences remain independent of the overall dataset size. This design choice guarantees that previously processed data remains unaffected when new samples are added and the project is reanalyzed. During denoising, Apscale automatically assigns unique identifiers to each sequence using the SHA3-256 hashing algorithm.

Several threshold types are available to control which reads are considered for denoising. By default, an absolute threshold is applied (minsize = 4), meaning that only sequences with an abundance of four or more are retained—effectively removing a substantial amount of low-abundance noise. Alternatively, a relative threshold can be used to retain only those sequences that represent a defined percentage of the sample’s total read count (e.g., 0.01%).

Since both absolute and relative thresholds are inherently arbitrary, we introduced a third option in version 4.0.0: power law–based filtering. Read abundance distributions typically follow a power law, where a few sequences are highly abundant (true biological signals) and many are rare (a mixture of real low-abundance taxa, sequencing noise, and PCR artifacts). This filtering method fits a power law model to each sample’s read distribution and sets the threshold at the point where the observed distribution deviates from the expected power law curve. The underlying assumption is that this inflection marks a shift in the signal-to-noise ratio, with noise becoming dominant. This approach results in a data-driven, rather than arbitrary, threshold for denoising.

To run this module, use the following command:

apscale --denoising [project\_path]

|  |  |  |  |
| --- | --- | --- | --- |
| Setting | Default | Possible values | Effect |
| perform denoising | True | True / False | Enables or disables the denoising module |
| alpha | 2 | 1 – inifity | alpha value for the denoising algorithm |
| treshold type | absolute | absolute / relative / power law | Filtering strategy to use to determine the abundance cutoff |
| size threshold [absolute nr / %] | 4 | 1 – infinity (absolute)  0 – infinity (relative, as %) | This can be set as either an absolute number (e.g., 4) or a percentage when using relative filtering. Power law–based filtering does not require a predefined threshold. |

## **Swarm clustering**

Alternatively, files can be clustered individually using the Swarm algorithm with d=1 and the fastidious option enabled by default (see Swarm [GitHub](https://github.com/torognes/swarm) for details). We consider Swarm clustering as an alternative to denoising; when retaining only the chosen centroid sequences, the results are generally quite similar. Additionally, the output from the denoising module can be further clustered with Swarm if desired.

|  |  |  |  |
| --- | --- | --- | --- |
| Setting | Default | Possible values | Effect |
| perform swarm clustering | False | True / False | Enables or disables the swarm clustering module |

## **Replicate merging**

This module merges replicates of the same sample, provided they follow a consistent naming pattern. It supports an unlimited number of replication levels (e.g., sample replicates, extraction replicates, PCR replicates). The replicate type should be indicated by a common delimiter in the file names and **must also be included in file names without replicates** to ensure reliable processing. See the example below.

To run this module, use the following command:

apscale --replicate\_merging [project\_path]

|  |  |  |  |
| --- | --- | --- | --- |
| Setting | Default | Possible values | Effect |
| perform replicate merging | True | True / False | Enables or disables the replicate merging module |
| replicate delimiter | \_ | Any character | Delimiter that indicates the replicate |
| minimum replicate presence | 2 | 1 to infinity | Minimum number of replicates in which a sequence must be present to be retained |

Example:

* Extraction replicates:

In our lab, we usually extract DNA from each sample twice and only retain sequences that are found in both replicates. Samples are named with suffixes indicating the extraction replicate, such as Sample1\_A and Sample1\_B, Sample2\_A and Sample2\_B, and so on. If we want to merge these replicates, the settings would be configured accordingly to keep only sequences present in both replicates:



* PCR replicates, extraction replicates, field replicates:

Consider a more complex scenario like eDNA sampling. Sometimes it takes two filters to process 1 liter of water, so some samples have one filter while others have two. This can be indicated in the sample name as sample1\_filter1, sample1\_filter2, etc. On top of that, extraction replicates are performed as described above, resulting in sample names like sample1\_filter1\_A and sample1\_filter1\_B. Additionally, there may be multiple PCR replicates for each extraction replicate, indicated by another delimiter. For example, with four PCR replicates per extraction replicate, sample names might look like this: sample1\_filter1\_A\_1, sample1\_filter1\_A\_2, sample1\_filter1\_A\_3, sample1\_filter1\_A\_4, sample1\_filter1\_B\_1, sample1\_filter1\_B\_2, sample1\_filter1\_B\_3, sample1\_filter1\_B\_4.

In the final analysis, depending on the research goal, you might only be interested in sequences from sample1, with all replicates properly merged. The filtering would be configured as follows:



First, keep any sequence present in at least one of the PCR replicates. Second, keep only sequences found in both extraction replicates. Finally, keep sequences found in at least one of the filters used to process the sample volume.

## **Negative control filtering**

This module automatically subtracts the maximum number of reads found in the negative controls from the corresponding sequences. Negative controls must be identifiable by a common prefix, which defaults to NC\_. Negative control filtering is typically performed after replicate merging, since replicate merging helps remove background noise introduced in the lab. Reads remaining in the negative controls after merging are considered potential true contaminants.

To run this module, use the following command:

apscale --nc\_removal [project\_path]

|  |  |  |  |
| --- | --- | --- | --- |
| Setting | Default | Possible values | Effect |
| perform nc removal | True | True / False | Enables or disables the nc removal module |
| negative control prefix | NC\_ | Any characters | Prefix that defines the negative controls |

## **Read table generation and sequence grouping**

This module generates the read table and performs threshold-based sequence grouping, similar to classical OTU clustering. Apscale always outputs both sequences (ESVs) and sequence groups (OTUs). The read table is saved in Parquet format and, if the dataset contains fewer than 1,000,000 distinct sequences, also in Excel format.

Additionally, this module creates a “read data store,” a DuckDB (https://duckdb.org/) database that contains comprehensive information about sequences, groups, samples, and read counts. The read data store efficiently handles even very large datasets—potentially billions of sequences—at high speed, without requiring the entire dataset to be loaded into memory. This makes it especially useful for scaling up analyses.

The read data store is extensively used in the Apscale analyze module, which is described in the following chapter.

To run this module, use the following command:

apscale --generate\_read\_table [project\_path]

|  |  |  |  |
| --- | --- | --- | --- |
| Setting | Default | Possible values | Effect |
| generate read table | True | True / False | Enables or disables the read table generation module |
| sequence group threshold | 0.97 | 0 - 1 | Defines the clustering threshold |

## **Working with the read data store**

The read data store enables users to perform a variety of useful operations on large datasets. Users can add metadata for samples and sequences, while the analyze module automatically infers corresponding metadata for sequence groups. Additionally, users can harmonize and validate taxonomy via GBIF, upload studies to ENA, and export read tables with custom filters.

## **Apscale analyze**

The Apscale analyze module is implemented via streamlit (<https://streamlit.io/>) which generates an interface in the browser to directly interact with the read data store.

To run this module, use the following command:

apscale --analyze [project\_path]

This will open the default browser and display the interface. An overview windows will give you basic information about the current project.

## **Adding sample metadata**

This module allows users to add sample metadata to the read data store. The input must be a table containing sample identifiers in one column and any number of additional columns with metadata (e.g. geographical location, habitat, sampler, sampling date, etc.). Metadata can be provided in Excel, Parquet, or CSV format and added via drag and drop.

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Example sample metadata:



Once the metadata has been uploaded, the module will automatically try to infer the columns and corresponding datatypes:

Ein Bild, das Text, Screenshot, Software, Design enthält.

KI-generierte Inhalte können fehlerhaft sein.

Currently Apscale supports strings, integers, floating point number, timestamps and Boolean values. One the sample metadata is set up accordingly, the button “Save to read data store” will add the data to the database. Once this is done, the module will display a preview of the metadata in the read data store.

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.

## **Adding sequence metadata**

This module allows users to add sequence metadata to the read data store. The input must be a table containing sample identifiers in one column and any number of additional columns with metadata (e.g. geographical location, habitat, sampler, sampling date, etc.). Metadata can be provided in Excel, Parquet, or CSV format and added via drag and drop. The logic is the same as with the sample metadata. Upon saving the metadata will be added to the read data store.

## **Correcting species names via GBIF**

This module can be used to harmonize species names with the GBIF taxonomic backbone, ensuring consistency and comparability across different studies. To run this module, the sequence metadata must include a column containing species names. The column can be selected via the dropdown menu and a preview will display all distinct species name found in this column.

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Once the button is clicked, Apscale will query the GBIF API and add a column named “gbif\_taxonomy” to the sequence metadata..

Ein Bild, das Text, Screenshot, Schrift, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

## **Validating species occurrence via GBIF**

This module validates taxonomy using GBIF occurrence data. It requires both GBIF taxonomy information and geographic coordinates (latitude and longitude) for each sample in the dataset. Apscale calculates a user-defined radius (default: 200 km) around the known occurrences of each sequence, which is then used to assess whether the taxonomic assignment is geographically plausible.

Ein Bild, das Text, Screenshot, Schrift, Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

Once the distributions are computed, species occurrences across all samples will be displayed on the map as red points, while the computed distribution ranges will be shown as green points.

Ein Bild, das Text, Karte, Screenshot, Diagramm enthält.

KI-generierte Inhalte können fehlerhaft sein.

Once the button is clicked, Apscale will validate all species names by checking whether any GBIF records for each species fall within the computed distribution range. The result of the validation procedure will be saved to the sequence metadata as either plausible or implausible in a column called “gbif\_validation”. After computation a preview will show the results.

Ein Bild, das Text, Screenshot, Software, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.

## **Uploading datasets to ENA**

This module will be added in version 4.2.

## **Exporting read tables**

The final module allows users to export and filter read tables. An unlimited number of filters can be applied to both sequence metadata and sample metadata. Additionally, output tables can be split based on sample metadata fields, enabling parallelized downstream analysis.

The results are exported to the project folder under 12\_analyze/data in both Parquet and CSV formats. Users can also select which columns to include in the output independently of any applied filters. Outputs will contain a table for sequences as well as for sequence groups.

Example:

Ein Bild, das Text, Screenshot, Schrift, Design enthält.

KI-generierte Inhalte können fehlerhaft sein.

In this example, the output tables will only contain samples where the Phylum matches Annelida or Arthropoda and the sampler is Dominik. Read tables will be split by the type column and include the selected sequence metadata columns.

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

The output folder will contain two files (.parquet, .csv) for sequences as well as the sequence groups.

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Apscale will create a log with the selected filters and one folder per “type” as the data is split by forest and river.

