pipemake: A pipeline creation tool using Snakemake for reproducible analysis of biological datasets

Andrew E. Webb,[1,2,3,*] **Scott W. Wolf**,[1,2,4] **Ian M. Traniello**[1,2] and **Sarah D. Kocher**[1,2,3]

[1]Lewis-Sigler Institute for Integrative Genomics, Princeton University, Princeton, NJ, USA, [2]Department of Ecology and Evolutionary Biology, Princeton University, Princeton, NJ, USA, [3]Howard Hughes Medical Institute, Chevy Chase, MD, USA and [4]Research Computing, Princeton University, Princeton, NJ, USA

*Corresponding author: aewebb@princeton.edu

Abstract

The exponential growth in biological data generation has created an urgent need for efficient, reproducible computational analysis workflows. Here, we present pipemake, a computational platform designed to streamline the development and implementation of efficient and reproducible Snakemake workflows. pipemake creates modular pipelines that can be seamlessly integrated or removed from the platform without requiring reconfiguration of the core system, enabling flexible adaptation of workflows to different analytical needs across diverse fields. To demonstrate the platform's capabilities, we created and implemented pipelines to reanalyze two distinct biological datasets. First, we recreated a population genomics analysis of the socially flexible halictid bee, *Lasioglossum albipes*, using pipemake-generated workflows for *de novo* genome annotation, processing of variant data, dimensionality reduction, and a genome-wide association study (GWAS). We then used pipemake to analyze behavioral tracking data from the common eastern bumble bee, *Bombus impatiens*. In both cases, pipemake workflows produced results consistent with published findings while substantially reducing hands-on analysis time. Overall, pipemake's modular design allows researchers to easily modify existing pipelines or develop new ones without software development expertise. Beyond streamlining workflow creation, pipemake leverages the full Snakemake ecosystem to enable parallel processing, automated error recovery, and comprehensive analysis documentation. These features make pipemake an efficient and accessible solution for analyzing complex biological datasets. pipemake is freely available as a conda package or direct download at `https://github.com/kocherlab/pipemake`

Introduction

The rate of big data production in biological science has rapidly increased over the past decades, leading to a discrepancy between the amount of data generated and our ability to meaningfully analyze it [1]. For example, advances in molecular biology have enabled high-throughput genomic data generation for nearly any organism, leading to a growing demand for genome-scale analyses in many different species. This has generated a significant need for approachable software and strategies to guarantee that findings are interpretable and reproducible [2, 3]. As such, there has been a massive influx of software packages designed by computational biologists targeted at addressing questions in the life sciences.

Much of the currently available software can be separated into three categories: toolkits, wrappers, and pipelines.

- **Toolkits** provide a collection of analytical tools that are typically designed to offer a variety of functions that operate on a specific file format and/or datatype (e.g. VCFtools, BCFtools, BEDTools, PLINK, GATK, SAMtools, etc.) [4, 5, 6, 7, 8]. While highly useful and invaluable in their area of expertise, they are unlikely to encompass the entirety of an analysis.
- **Wrappers** are software designed to call other libraries or executables (e.g. admixr, Funannotate, Galaxy, etc.) [9, 10, 11]. Often they involve simplifying the user interface and/or enabling the integration of software across multiple programming languages. Wrappers are typically used to perform a single task or command rather than a complete pipeline. While software wrappers are highly useful, they often lack features of the wrapped software and possess limitations imposed by method/language used to call the software.
- Self-contained **pipelines** have become more prevalent since the advent of conda environments and docker containers. They typically operate by automatically invoking commands from a collection of tools/software from different sources and/or programming languages (e.g. BRAKER, insectOR, etc.) [12, 13]. While such pipelines have the advantage of being easy to use, they may also act as a "black box" to most users who do not know the commands, their significance, and/or the order in which they operate. Additionally, the operations of contained pipelines are typically fixed and cannot be easily edited or repurposed.

Modern analyses typically require a combination of these different types of tools to operate. These tools are often combined by two key approaches. The first and far more common approach is to create lists of commands that call the necessary software. This method is inherently limiting due to hard-coded filenames and arguments that quickly make long lists of commands impractical to modify or operate with larger datasets and more sophisticated pipelines. These limitations frequently lead to the second approach, in which the software is repackaged within a new wrapper or pipeline. While the latter approach is often tempting, it may yield an overly complicated command structure and potentially limited, inefficient, and/or confusing resource requirements.

The limitations of both approaches can be overcome by using a workflow management system, such as Snakemake [14]. Snakemake creates workflows using short, text-based files that contain a standardized rule system. Each rule represents a single procedure within a workflow, and rules will be automatically invoked in a user-defined order to produce the desired output. Rules may vary in required resources such as threads and memory, file formats/types, and even software versions, all of which are customizable by the user. Rules may also be run in parallel, expediting workflows by fully utilizing High Performance Computing (HPC) environments or even systems with higher core counts. Snakemake also offers an ecosystem that is easily configurable, editable, and more importantly, reproducible.

While Snakemake provides an attractive ecosystem, some issues remain. First, developers are required to learn the rule nomenclature, which may present a challenge to those unfamiliar with programming. Additionally, reconfiguring and/or repurposing Snakemake rules can be unclear and error-prone for inexperienced developers. For researchers who are only interested in running a standalone analysis, such obstacles may result in Snakemake being an impractical solution.

For users more comfortable within the Snakemake ecosystem, some inconveniences still remain. Configuration files (”configs”) mitigate some of the issues in reconfiguring Snakemake workflows, but there is no system to rapidly generate or modify configs beyond editing previously created ones. While this is not inherently problematic, it represents a substantial investment of time, is highly error-prone, and is more likely to result in loss or corruption of the original config. Snakemake files may be reused across workflows, but this process is similarly subject to coding errors, especially if similar files with key parameter changes exist in a user’s file system. Moreover, workflow integrity is compromised without robust record-keeping and easy access to file modification history. This creates a need for a straightforward means of implementing a workflow manager, like Snakemake, within a platform that transparently manages the more difficult aspects of workflow generation, modification, and execution.

To this end, we present pipemake as a solution to reduce the barrier for users to generate and run reproducible computational analyses in the Snakemake ecosystem. pipemake is a pipeline creation tool designed to facilitate the development of Snakemake workflows to optimize computational efficiency and reproducibility. The workflows produced with pipemake can interface effectively with high performance computing systems to handle resource allocation using Snakemake’s built-in capabilities. Thus, pipemake enables the parallel processing of tasks and automatic determination of the optimal execution order. Moreover, workflows may be automatically resumed from a point of failure without a need to rerun previously completed steps.

pipemake was designed to streamline pipeline development by creating a flexible, modular platform with swappable pipelines that can easily reuse previously written Snakemake code. The package includes a collection of curated, customizable genomic and behavioral analysis pipelines for researchers seeking to rapidly integrate Snakemake-based workflows into their research, and additional modules can easily be integrated into the platform. Instead of a collection of immutable pipelines, pipemake is a unique platform that provides a flexible system for simplifying the creation and modification of Snakemake workflows. This nature allows pipemake to support a larger community of researchers than if it concentrated on a single aspect alone. Our hope is to grow this community as more snakemake workflows are developed, especially from different fields of study.

To demonstrate the broad utility of pipemake, we replicated computational analyses from previous work in two distinct fields of study: a population genomics study of social behavior in a socially flexible sweat bee, *Lasioglossum albipes*, and an automated behavioral tracking experiment in the common eastern bumble bee, *Bombus impatiens* [15, 16]. The pipelines we developed successfully replicated findings from both studies, and since they were developed within the pipemake platform, they may be easily reused or repurposed for independent datasets.

pipemake

Design

pipemake was designed to operate using two primary filetypes that we outline in the following sections: snakemake files and a pipemake config file.

Snakemake files

pipemake uses standard Snakemake files, which maintain their primary function of including the necessary rule(s) for a procedure or analysis [14]. The only requirement imposed by pipemake is that all paths begin with a pipemake-specific configurable directory to ensure that files are correctly assigned. pipemake is also capable of using Snakemake files to determine the necessary requirements for configuration (i.e. parameters, containers, resources, paths, etc.) and confirm all requirements were provided by the user. While pipemake operates on standard Snakemake files, ideally they should be modular in function and have configurable paths and parameters. The primary benefit of modular Snakemake files within the pipemake platform is that they can be easily reused, with the only requirement being the consistent specification of filenames and directories (Fig. 1).

Snakemake rules are also capable of operating within a conda environment or a singularity container [17, 18]. We chose to limit Snakemake rules in pipemake to singularity containers in order to reduce complexity and prioritize the benefits of providing all required software in container format. A key benefit of containers is that they enable software to operate within any operating system, a critical feature for maintaining the compatibility of the pipemake platform when adding future pipelines. Moreover, Snakemake automatically downloads and builds containers, eliminating the need for users to maintain software themselves [19].

To take full advantage of this functionality, we have created containers for all software currently required by our pipelines, and we will add more containers as new official pipelines are released. We have also designed pipemake to allow users to define a directory in which containers are stored. This option was added for groups who wish to have one set of containers that are shared by multiple users.

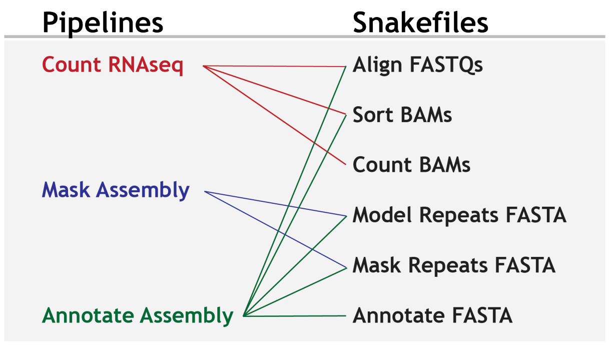


Fig. 1: **Example pipelines and Snakemake files** pipemake uses a simple system to associate Snakemake files with pipelines allowing them to be reused where appropriate.

Pipeline Configuration files

pipemake configuration files are YAML-based and are used by pipemake to define pipelines and provide: command-line arguments, the procedure to standardize input for the workflow, and a list of the required Snakemake files (Fig. 2). This is the primary file that users will need to define their own pipelines.

To define a pipeline, the only requirements are i) a unique pipeline name, ii) a description of the pipeline, and iii) the pipeline version. The complexity of the command-line arguments and input standardization is mostly dependent on the desired configurability of a pipeline. At a minimum, only the file input requires a command-line argument and a standardization procedure. Command-line arguments are parsed using the argparse python library and follow their standard nomenclature with some limitations to minimize compatibility issues. Standardization procedures are primarily used to ensure the input is correctly assigned for a procedure. Files are standardized by creating a copy or symbolic link of the file with a standardized name. It is also possible to have a standardization processes within a Snakemake file, such as providing a collection of sequence read archive (SRA) IDs which are standardized by directly downloading the files from the database [20]. And lastly, snakemake files are associated with a pipeline by including their filename and may be added or removed as needed.

A typical pipemake workflow begins by calling the executable, which will produce a list of the available pipelines along with a description of their function. Calling pipemake followed by the name of a pipeline will automatically produce the help text, listing all the relevant command-line arguments for the pipeline.

For most pipelines, the only command-line arguments required are the input (e.g. files, directories, SRA IDs, etc.) and critical parameters. When pipemake is called with a pipeline and the required command-line arguments, a workflow directory is produced along with the necessary workflow Snakemake files (Fig. 3).

The workflow directory contains the pipemake directory and the standardized input or symbolic links, if not being downloaded from a database by the pipeline. The pipemake directory is used to store the pipeline Snakemake files, log files, and copies of the workflow Snakemake files. The workflow Snakemake file is used to define the desired output and links to the Snakemake files within the pipemake directory. A configuration file is also produced which defines all the required configurables alongside the default resources for each rule in the pipeline.

To start an analysis, the user is only required to execute Snakemake with the workflow Snakemake file. This allows users the flexibility of performing the analysis as desired - i.e. single-core, multi-core, multi-job on HPCs, etc.

pipemake requires both the Snakemake files and pipeline configuration files to be stored in a predefined, configurable location. Additional pipelines may be added to pipemake simply by storing a new pipeline configuration file alongside any necessary Snakemake files. Once stored, pipemake will be capable of operating the new pipeline.

Several pipelines are included in the current release that enable a range of data analyses, including applications for: *de novo* genome annotations, the analysis of transcriptomic and chromatin accessibility data, population genomic data, and automated behavioral tracking data (see Table 1 for complete list). For detailed installation instructions and an example of pipemake implementation, please see the full documentation: https://pipemake.readthedocs.io/en/latest/.

Case study 1: pipemake implementation of *de novo* genome annotation

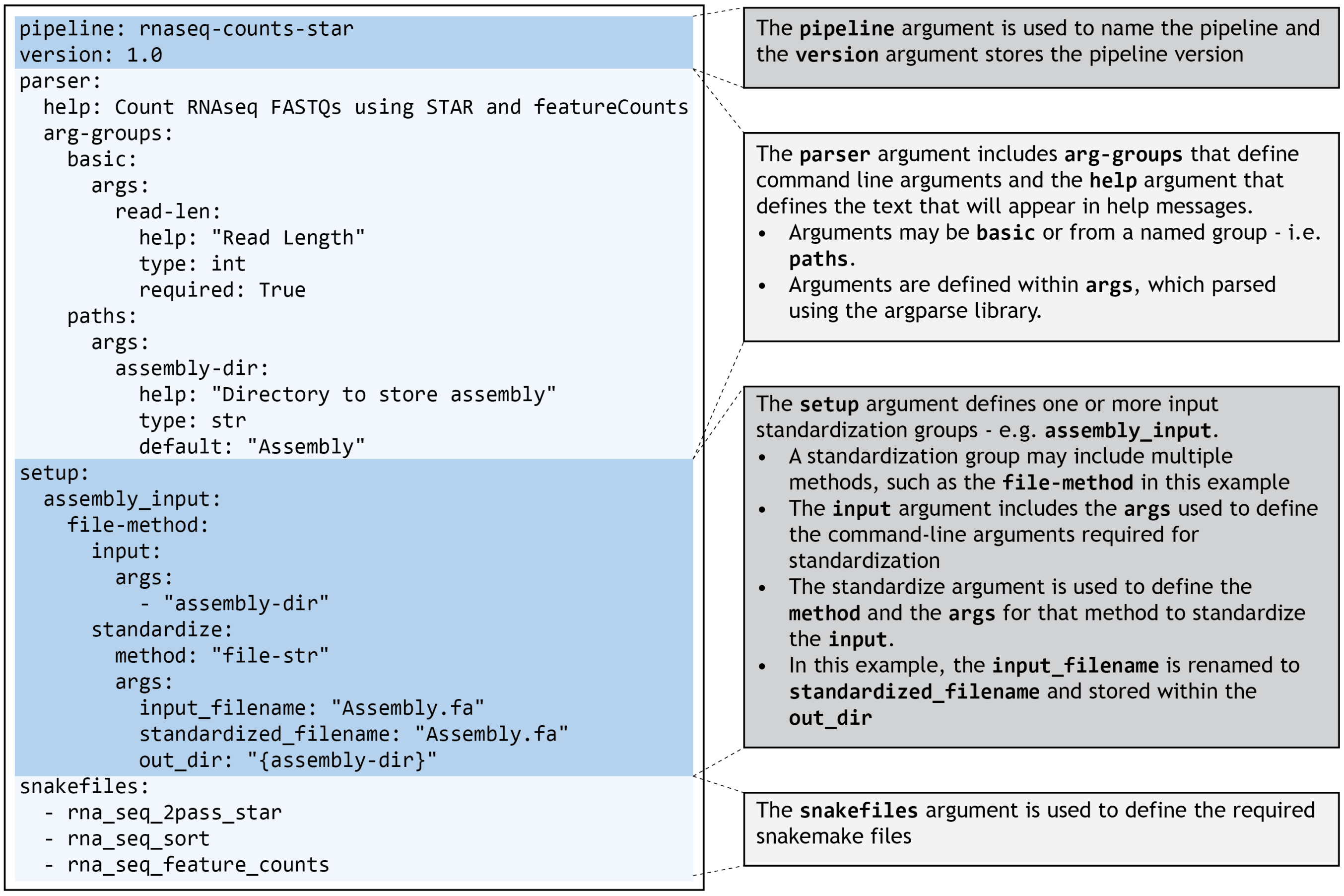
pipemake's short-read annotation pipeline, `annotate-braker3`, operates using BRAKER3, HISAT2, and RepeatModeler [13, 21, 22, 23]. We evaluated the pipeline using a recently released chromosomal-level assembly of *Lasioglossum albipes*, 40 paired-end RNAseq samples (NCBI SRA accession: PRJNA1142947), and a homology database constructed from OrthoDB and the OMA Orthology database [24, 25, 26]. Our snakemake workflow completed in approximately 38 hours running on a maximum 120 cores (or 20 jobs) without any intervention.

The pipeline resulted in 10,979 genes with a BUSCO score of 97.7%. This is fewer than the 15,905 genes inferred from a previous, more fragmented assembly [15]. Therefore, this discrepancy is likely partially due to less fragmented gene models, which might also explain the lower BUSCO score of 95.6% of the previous annotations.

We found the `annotate-braker3` pipeline to be simple to operate and efficient with minimal investment by the user. The pipeline was also simple to scale, allowing for the number of samples to have minimal effect on the overall runtime. These strengths make `annotate-braker3` an ideal method for exploratory studies and a foundation from which more sophisticated methods could be developed.

Case study 2: pipemake can analyze population genetic data

To evaluate the ability of pipemake's population genomics pipelines to streamline an analysis, we sought to replicate results from a previously population genomic study comparing social and solitary populations of a socially flexible sweat bee, *Lasioglossum albipes* [15] (NCBI PRJNA413432). Since the original publication, an improved chromosomal-level assembly has been released [24]. We used the updated assembly described above and previously published whole-genome resequencing data from [15], which we mapped to the new assembly.



```
pipeline: rnaseq-counts-star
version: 1.0
parser:
  help: Count RNAseq FASTQs using STAR and featureCounts
  arg-groups:
    basic:
      args:
        read-len:
          help: "Read Length"
          type: int
          required: True
    paths:
      args:
        assembly-dir:
          help: "Directory to store assembly"
          type: str
          default: "Assembly"
setup:
  assembly_input:
    file-method:
      input:
        args:
          - "assembly-dir"
      standardize:
        method: "file-str"
        args:
          input_filename: "Assembly.fa"
          standardized_filename: "Assembly.fa"
          out_dir: "{assembly-dir}"
snakefiles:
  - rna_seq_2pass_star
  - rna_seq_sort
  - rna_seq_feature_counts
```

The **pipeline** argument is used to name the pipeline and the **version** argument stores the pipeline version

The **parser** argument includes **arg-groups** that define command line arguments and the **help** argument that defines the text that will appear in help messages.
- Arguments may be **basic** or from a named group - i.e. **paths**.
- Arguments are defined within **args**, which parsed using the argparse library.

The **setup** argument defines one or more input standardization groups - e.g. **assembly_input**.
- A standardization group may include multiple methods, such as the **file-method** in this example
- The **input** argument includes the **args** used to define the command-line arguments required for standardization
- The standardize argument is used to define the **method** and the **args** for that method to standardize the **input**.
- In this example, the **input_filename** is renamed to **standardized_filename** and stored within the **out_dir**

The **snakefiles** argument is used to define the required snakemake files

Fig. 2: **Structure of a Pipeline Configuration File**. pipemake pipeline configuration files are separated into four categories: pipeline assignment, parser arguments, input standardization procedure, and Snakemake file requirements.

We used the snpArcher platform [27] to create the variant calls and generate the raw VCF file used in subsequent analyses. We next used the pipemake pipeline `filter-model-vcf` (Table 1) to filter the VCF file to 139 samples in our model using bcftools [7]. By default, `filter-model-vcf` will filter an input VCF/BCF in two steps: i) remove samples not specified within the model and ii) only include biallelic SNPs that pass the specified thresholds for MAF, quality, and missingness. pipemake also offers pipelines that will filter on all samples or on multiple populations/species. After filtration, we had a total of 4,159,171 SNPs; this is considerably higher than reported in Kocher et al. 2018 but not surprising considering we used a higher quality assembly with fewer gaps as well as a joint genotyping pipeline [15].

We next used the pipemake pipeline `reseq-popgen` (Table 1) to perform a basic set of analyses on the filtered VCF using bcftools, plink, and GEMMA [6, 7, 28]. This pipeline includes options for calculating Fst, LD-pruning the VCF, and performing a PCA and GWAS on the pruned dataset (Fig. 4A) (see Supplemental Methods for additional details). Our snakemake workflow completed in approximately 7 minutes running on a maximum 200 cores (or 60 jobs) without any intervention.

Overall, we found a large degree of overlap between Kocher et al 2018 and our reanalysis with pipemake. The PCA identified similar patterns of population clustering to Kocher et al. 2018, with the only exception being the separation of two samples from the VOS population from a cluster of samples from the BRS population (Fig. 4B). This is likely due to our use of a higher quality assembly and/or the improved genotyping calls from the updated joint genotyping pipeline. Moving on to the GWAS analysis, our pipeline replicated the peak at the location of *syntaxin 1A* (*syx1a*) (Fig. 4C). Closer examination of the peak found the seven variants, three within the first intron and four upstream, the same as reported in Kocher et al. 2018. In addition to the peak associated with *syx1a*, we also identified new peaks that were not observed in the original analysis and genome assembly (Fig. S1). Some of these peaks were found among odorant receptors, which suggests additional targets for future investigation. Overall, these findings illustrate how pipemake can successfully recapitulate previously published pipelines and results.

Case study 3: pipemake can perform automated behavioral tracking using NAPS

To demonstrate the versatility of the pipemake platform and its application beyond genomic datasets, we built and deployed a pipeline to replicate findings from a behavioral study of the common eastern bumble bee *Bombus impatiens*, recreating an analysis performed by NAPS (NAPS is ArUco Plus SLEAP), an automated behavioral monitoring software [16]. Specifically, NAPS utilizes hybrid tracking, or the coupling of a fiducial marker (i.e.,

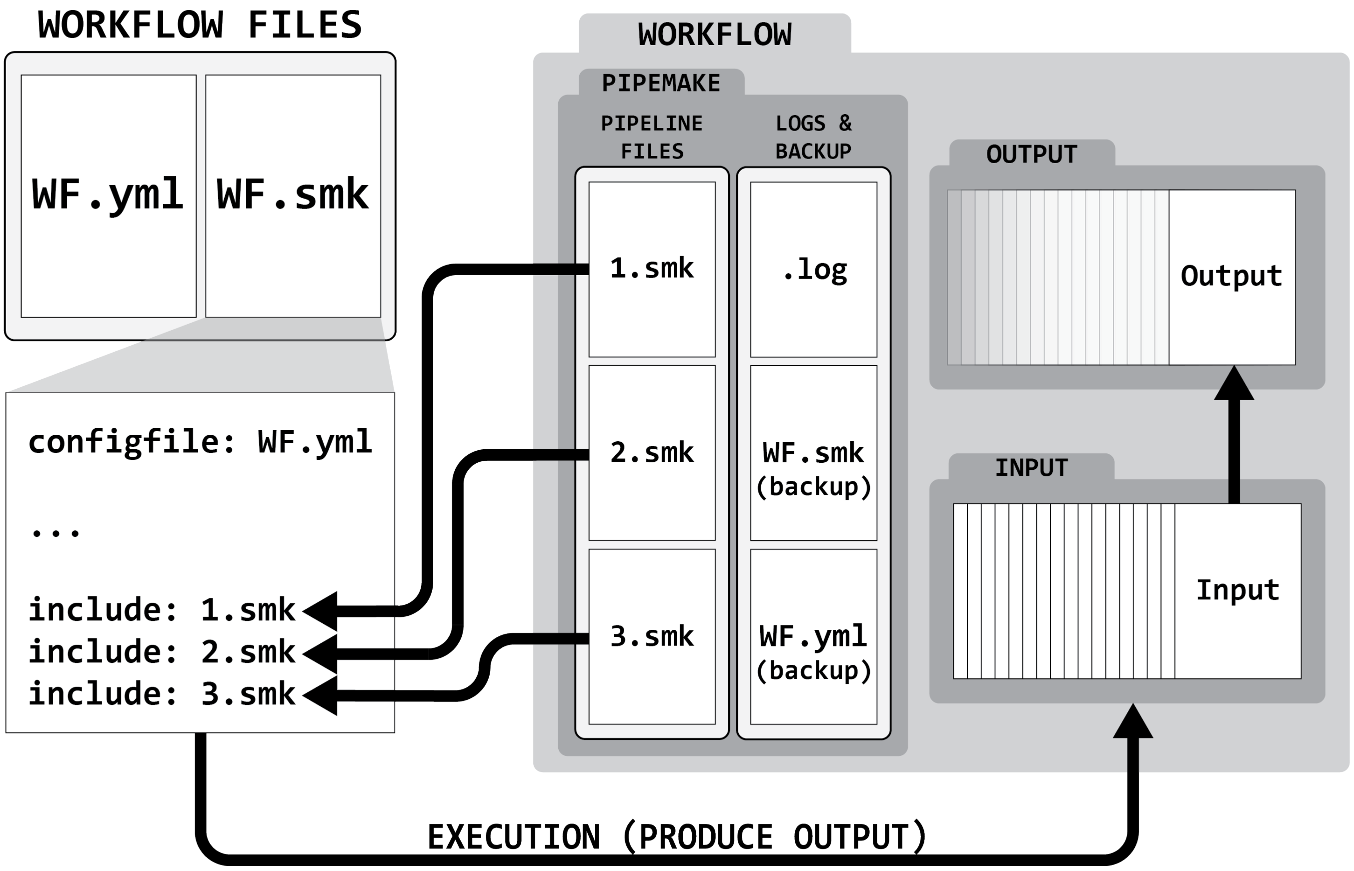


Fig. 3: **Structure of a pipemake Workflow**. A pipemake workflow includes a directory, a Snakemake file, and a configuration file. The workflow Snakemake file is designed to be a single file that loads the config yml (WF.yml) and aggregates the rules defined within the pipeline Snakemake files (WF.smk). The pipeline Snakemake files (1.smk, 2.smk, 3.smk) are stored within the workflow directory and contain the rules for the workflow to function. When the workflow Snakemake file is executed it will produce the desired output within the workflow directory. The purpose of this schema is keep a record of all files required by the workflow to simplify record keeping and reproducibility. For this reason, pipemake automatically stores a backup of the workflow Snakemake and configuration files. Lastly, pipemake records all command-line arguments and file processing steps within a log file.

a barcode) with pose estimation, a process in which a computer is tasked with recognizing an animal's position in space [29, 30]. With NAPS, an investigator can monitor and quantify physical behaviors while also retaining individual identity over the course of an experiment.

The original NAPS workflow in Wolf et al. (2023) required video decoding, pose estimation, identity detection, and various post-processing steps to be run in sequence with frequent command-line intervention from the user. Using pipemake, these tasks were run in parallel and within a contained workflow that required minimal user intervention upon initiation, and our reanalysis perfectly matched initial findings: while either temporal tracking with SLEAP or hybrid tracking with NAPS identified nearly every individual in a video frame (Fig. 5, left), only NAPS correctly estimated the number of unique individuals filmed over the course of the tracked experiment (Fig. 5, right). Behavioral experiments involving different organisms or filming setups can reuse or adapt this workflow by swapping components like pose estimation models or behavioral analysis modules to best fit the focal system.

Discussion

pipemake streamlines and expedites the development of transparent and reproducible Snakemake workflows for users with a wide range of programming experience. pipemake may be applied to any analysis pipeline requiring multi-step data processing workflows. pipemake's modular structure makes it easy to extend or adapt workflows, allowing researchers to incorporate or modify existing pipelines without significant reconfiguration.

pipemake provides an accessible platform that can be used across a wide range of disciplines and input datasets. Here, we demonstrate cross-discipline generalizability by reanalyzing both genomic data as well as automated behavioral tracking data through a pipemake implementation of each dataset's associated computational platforms. In both cases, pipemake replicated our previously published results, demonstrating its utility in building reproducible scientific data analysis.

The most obvious benefit of using pipemake is the ease of performing the same analysis on multiple species, across independent datasets, and in independent directories by only needing to change the relevant input arguments. For example,

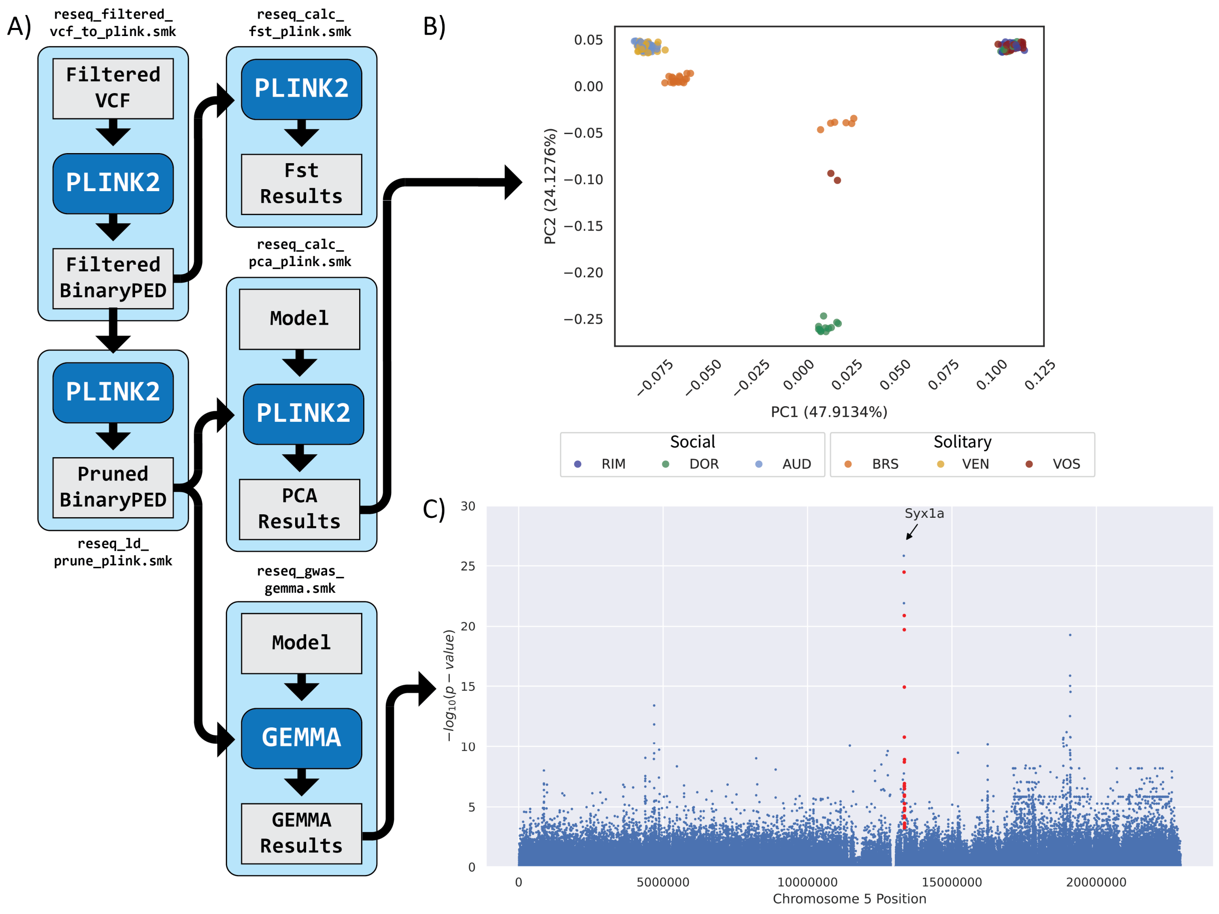


Fig. 4: **pipemake workflow for implementing Fst, principal component analysis (PCA), and a genome-wide association study (GWAS) using example data from the socially polymorphic sweat bee, *Lasioglossum albipes*.** A) pipemake can implement genome-wide data analyses, such as the genome-wide PCA and GWAS using the `reseq-popgen` pipeline. The diagram depicts the primary snakemake files (light blue boxes) and the order in which they are executed to perform the analyses. Each snakemake file indicates the required input (light gray), the primary program (dark blue), and the output (light gray). Arrows indicate the flow from input to output, including output files that become input for subsequent snakemake files. Please note that not all steps are given for each snakemake rule for clarity. B) The PCA implemented by `reseq-popgen` replicates observations from [15]. For clarity, we have included a jitter to make each population visible among the clusters. C) The GWAS implemented by identified a similar peak associated with variation in social behavior in the sweat bee, *Lasioglossum albipes*. SNPs associated with social behavior and their genomic coordinates are located on chromosome 5 in an updated genome assembly. Each point in the manhattan plot represents a single SNP and its $-\log_{10}$ p-value. SNPs within 10kb of Syx1a and a FDR <0.2 are shown in red.

in comparative genomics, users often create a new directory that implements an identical file structure for each species of interest; with pipemake, a user may perform the same comparative analyses by simply modifying the name and location of the data input directory on the command-line.

Modifying existing pipelines and developing new ones is greatly simplified given pipemake's modular nature. New pipelines can be readily created using a combination of existing Snakemake modules and new ones. For instance, the creation of our genome annotation pipeline was greatly streamlined as we integrated previously developed Snakemake modules for RNAseq alignment and repeat masking. Additionally, the creation of new pipeline configuration files is simplified by inserting the relevant components from previously created pipelines that use the same Snakemake files.

An unexpected benefit of pipemake's design was the the ease of adding quality control (QC) modules to existing pipelines. From our experience, once a QC module was designed for a particular datatype or procedure it could easily be propagated to all relevant pipelines. Even rule-specific or unique QC modules could be readily added to a pipeline without the need to alter other Snakemake files.

Implementing pipelines and troubleshooting is also streamlined within the pipemake platform. Using modular Snakemake files often allows errors to be easily attributed to a single rule, this is especially true when developing from an existing pipeline. Errors due to insufficient resource allocations can be resolved using pipemake's built-in resource-scalars for widespread issues or modifying the workflow configuration file for rule-specific issues. These same tools were also useful to adjust resources to better

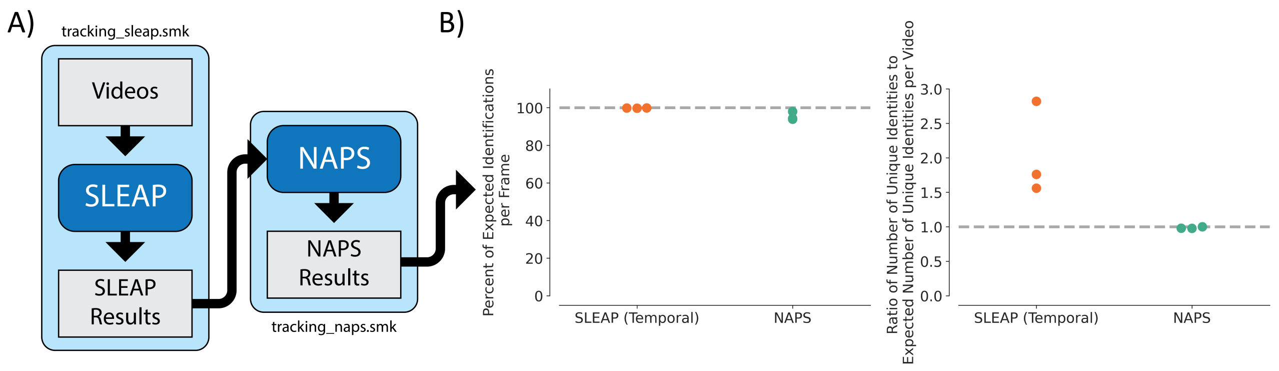


Fig. 5: **A comparison of the percent of expected identifications per frame between SLEAP an NAPS in an example data set.** For our data set, we expect all 50 bees to be identified by SLEAP. For NAPS and ArUco, we expect that all active individuals, where the ArUco tag should be detected at some point throughout the video, to be detected. This number varies from 44 to 48 across three videos in the data set. (right) A comparison of the ratio of the expected number of unique identities compared to the expected number of unique identities per video. We expect 50 unique identities for SLEAP and from 44 to 48 for NAPS based on the number of active individuals in each video. These analyses were implemented using the `tracking-naps` pipeline in pipemake.

utilize parallel processing and the larger RAM allocations allowed on HPC clusters. Also, if desired, the workflow directory allows for pipeline modifications to be explored without impact to the original pipeline.

The workflow directory is an ideal target for data preservation, as it stores all relevant pipeline files, the input, and the output. Preserving the workflow directory also provides other critical benefits. The directory may be used to produce a detailed report on all commands performed within a pipeline alongside a visual representation of the pipeline. This greatly simplified our record-keeping and allows for the records to be easily recreated if lost. The directory may also be used to easily rerun a pipeline, greatly simplifying the process of replicating an analysis.

The primary goal our pipeline management system is to simplify the process of maintaining and obtaining pipelines. Future updates to pipemake will primarily focus on feature improvements that facilitate pipeline development and availability. We will begin by developing a graphical user interface (GUI) to simplify pipeline creation and reduce the barriers to creating pipeline configuration files. We are also exploring the practicality of implementing an online database to store and maintain pipelines.

pipemake's modular nature and its ability to handle complex workflows make it an ideal platform to rapidly develop Snakemake workflows. Workflows designed in pipemake are scalable, easily reproducible, and streamlined for record-keeping and data preservation. Here, we have demonstrated pipemake's potential by replicating analyses in two distinct disciplines of biology, however, the platform can be expanded to any application that can operate within a workflow. pipemake is freely available as a conda package or direct download at `https://github.com/kocherlab/pipemake`.

Competing interests

No competing interest is declared.

Author contributions statement

AEW conceived of and wrote pipemake with input from SDK. SWW and IMT contributed pipemake modules and provided feedback on the software. AEW and SWW conducted data analyses. AEW wrote the original manuscript draft and received feedback from all co-authors.

Acknowledgments

The authors thank Brian Arnold, Sarthok Rahman, and other members of the Kocher Lab for beta testing and input. This work is supported in part by funds from the National Institutes of Health (NIH DP2 GM137424), and by the Packard Foundation and Pew Biomedical Scholars Program. IMT is a Lewis-Sigler Scholar and SDK is an HHMI Freeman Hrabowski Scholar.

References

1. Zachary D. Stephens, Skylar Y. Lee, Faraz Faghri, Roy H. Campbell, Chengxiang Zhai, Miles J. Efron, Ravishankar Iyer, Michael C. Schatz, Saurabh Sinha, and Gene E. Robinson. Big Data: Astronomical or Genomical? *PLOS Biology*, 13(7): e1002195, July 2015. ISSN 1545-7885. doi: 10.1371/journal.pbio.1002195. URL `https://dx.plos.org/10.1371/journal.pbio.1002195`.
2. Geir Kjetil Sandve, Anton Nekrutenko, James Taylor, and Eivind Hovig. Ten Simple Rules for Reproducible Computational Research. *PLoS Computational Biology*, 9 (10):e1003285, October 2013. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1003285. URL `https://dx.plos.org/10.1371/journal.pcbi.1003285`.
3. Jason A. Papin, Feilim Mac Gabhann, Herbert M. Sauro, David Nickerson, and Anand Rampadarath. Improving reproducibility in computational biology research. *PLOS Computational Biology*, 16(5):e1007881, May 2020. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1007881. URL `https://dx.plos.org/10.1371/journal.pcbi.1007881`.

Pipeline	Function
annotate-braker3	Annotate an assembly using BRAKER3
annotate-braker3-with-bam	Annotate an assembly using BRAKER3 using a processed BAM file
annotate-genes-eggnog	Annotate genes using eggNOG
annotate-utrs-peaks2utr	Annotate an 3prime-UTRs using peaks2utr using a merged BAM file
atacseq-peaks-macs3	Generate peaks for bulk ATAC-Seq data using bwa and MACS3
codon-alignments	Create codon alignments from untransleted CDS multiple sequence files
fastq-filter	Filter FASTQ files using fastp
filter-model-vcf	Filter a model in resequencing VCF files using bcftools
filter-pops-vcf	Filter populations in resequencing VCF files using bcftools
filter-vcf	Filter resequencing VCF files using bcftools
mask-assembly	Annotate an assembly using BRAKER3
reseq-calc-nsl	Calculate nSL using selscan
reseq-calc-xpnsl	Calculate XP-nSL using selscan
reseq-popgen	Run basic PopGen analyses on resequencing data
rnaseq-counts-star	Count RNAseq reads within a genome assembly using STAR and featureCounts
seperate-pop-vcfs	Create separate VCF files for each population in a resequencing VCF file
tracking-naps	Track MP4 videos using SLEAP and NAPS
unmapped-fastqs-star	Create unmapped FASTQs using STAR

Table 1. Current pipelines in pipemake

4. Aaron R. Quinlan and Ira M. Hall. BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6):841–842, March 2010. ISSN 1367-4811, 1367-4803. doi: 10.1093/bioinformatics/btq033. URL https://academic.oup.com/bioinformatics/article/26/6/841/244688.
5. Petr Danecek, Adam Auton, Goncalo Abecasis, Cornelis A. Albers, Eric Banks, Mark A. DePristo, Robert E. Handsaker, Gerton Lunter, Gabor T. Marth, Stephen T. Sherry, Gilean McVean, Richard Durbin, and 1000 Genomes Project Analysis Group. The variant call format and VCFtools. *Bioinformatics*, 27(15):2156–2158, August 2011. ISSN 1367-4803. doi: 10.1093/bioinformatics/btr330. URL https://doi.org/10.1093/bioinformatics/btr330.
6. Shaun Purcell, Benjamin Neale, Kathe Todd-Brown, Lori Thomas, Manuel A.R. Ferreira, David Bender, Julian Maller, Pamela Sklar, Paul I.W. De Bakker, Mark J. Daly, and Pak C. Sham. PLINK: A Tool Set for Whole-Genome Association and Population-Based Linkage Analyses. *The American Journal of Human Genetics*, 81(3):559–575, September 2007. ISSN 00029297. doi: 10.1086/519795. URL https://linkinghub.elsevier.com/retrieve/pii/S0002929707613524.
7. Petr Danecek, James K Bonfield, Jennifer Liddle, John Marshall, Valeriu Ohan, Martin O Pollard, Andrew Whitwham, Thomas Keane, Shane A McCarthy, Robert M Davies, and Heng Li. Twelve years of SAMtools and BCFtools. *GigaScience*, 10(2):giab008, February 2021. ISSN 2047-217X. doi: 10.1093/gigascience/giab008. URL https://doi.org/10.1093/gigascience/giab008.
8. Ryan Poplin, Valentin Ruano-Rubio, Mark A. DePristo, Tim J. Fennell, Mauricio O. Carneiro, Geraldine A. Van Der Auwera, David E. Kling, Laura D. Gauthier, Ami Levy-Moonshine, David Roazen, Khalid Shakir, Joel Thibault, Sheila Chandran, Chris Whelan, Monkol Lek, Stacey Gabriel, Mark J Daly, Ben Neale, Daniel G. MacArthur, and Eric Banks. Scaling accurate genetic variant discovery to tens of thousands of samples, November 2017. URL http://biorxiv.org/lookup/doi/10.1101/201178.
9. Martin Petr, Benjamin Vernot, and Janet Kelso. *admixr*—R package for reproducible analyses using ADMIXTOOLS. *Bioinformatics*, 35(17):3194–3195, September 2019. ISSN 1367-4803, 1367-4811. doi: 10.1093/bioinformatics/btz030. URL https://academic.oup.com/bioinformatics/article/35/17/3194/5298728.
10. Jonathan M. Palmer and Jason Stajich. Funannotate v1.8.1: Eukaryotic genome annotation, September 2020. URL https://zenodo.org/record/4054262.
11. The Galaxy Community, Linelle Ann L Abueg, Enis Afgan, Olivier Allart, Ahmed H Awan, Wendi A Bacon, Dannon Baker, Madeline Bassetti, Bérénice Batut, Matthias Bernt, Daniel Blankenberg, Aureliano Bombarely, Anthony Bretaudeau, Catherine J Bromhead, Melissa L Burke, Patrick K Capon, Martin Čech, María Chavero-Díez, John M Chilton, Tyler J Collins, Frederik Coppens, Nate Coraor, Gianmauro Cuccuru, Fabio Cumbo, John Davis, Paul F De Geest, Willem De Koning, Martin Demko, Assunta DeSanto, José Manuel Domínguez Begines, Maria A Doyle, Bert Droesbeke, Anika Erxleben-Eggenhofer, Melanie C Föll, Giulio Formenti, Anne Fouilloux, Rendani Gangazhe, Tanguy Genthon, Jeremy Goecks, Alejandra N Gonzalez Beltran, Nuwan A Goonasekera, Nadia Goué, Timothy J Griffin, Björn A Grüning, Aysam Guerler, Sveinung Gundersen, Ove Johan Ragnar Gustafsson, Christina Hall, Thomas W Harrop, Helge Hecht, Alireza Heidari, Tillman Heisner, Florian Heyl, Saskia Hiltemann, Hans-Rudolf Hotz, Cameron J Hyde, Pratik D Jagtap, Julia Jakiela, James E Johnson, Jayadev Joshi, Marie Jossé, Khaled Jum'ah, Matúš Kalaš, Katarzyna Kamieniecka, Tunc Kayikcioglu, Markus Konkol, Leonid Kostrykin, Natalie Kucher, Anup Kumar, Mira Kuntz, Delphine Lariviere, Ross Lazarus, Yvan Le Bras, Gildas Le Corguillé, Justin Lee, Simone Leo, Leandro Liborio, Romane Libouban, David López Tabernero, Lucille Lopez-Delisle, Laila S Los, Alexandru Mahmoud, Igor Makunin, Pierre Marin, Subina Mehta, Winnie Mok, Pablo A Moreno, François Morier-Genoud, Stephen Mosher, Teresa Müller,

Engy Nasr, Anton Nekrutenko, Tiffanie M Nelson, Asime J Oba, Alexander Ostrovsky, Polina V Polunina, Krzysztof Poterlowicz, Elliott J Price, Gareth R Price, Helena Rasche, Bryan Raubenolt, Coline Royaux, Luke Sargent, Michelle T Savage, Volodymyr Savchenko, Denys Savchenko, Michael C Schatz, Pauline Seguineau, Beatriz Serrano-Solano, Nicola Soranzo, Sanjay Kumar Srikakulam, Keith Suderman, Anna E Syme, Marco Antonio Tangaro, Jonathan A Tedds, Mehmet Tekman, Wai Cheng (Mike) Thang, Anil S Thanki, Michael Uhl, Marius Van Den Beek, Deepti Varshney, Jenn Vessio, Pavankumar Videm, Greg Von Kuster, Gregory R Watson, Natalie Whitaker-Allen, Uwe Winter, Martin Wolstencroft, Federico Zambelli, Paul Zierep, and Rand Zoabi. The Galaxy platform for accessible, reproducible, and collaborative data analyses: 2024 update. *Nucleic Acids Research*, 52(W1): W83–W94, July 2024. ISSN 0305-1048, 1362-4962. doi: 10.1093/nar/gkae410. URL `https://academic.oup.com/nar/article/52/W1/W83/7676834`.

12. Snehal Dilip Karpe, Vikas Tiwari, and Sowdhamini Ramanathan. InsectOR—Webserver for sensitive identification of insect olfactory receptor genes from non-model genomes. *PLOS ONE*, 16(1):e0245324, January 2021. ISSN 1932-6203. doi: 10.1371/journal.pone.0245324. URL `https://dx.plos.org/10.1371/journal.pone.0245324`.
13. Lars Gabriel, Tomáš Brůna, Katharina J. Hoff, Matthis Ebel, Alexandre Lomsadze, Mark Borodovsky, and Mario Stanke. BRAKER3: Fully automated genome annotation using RNA-seq and protein evidence with GeneMark-ETP, AUGUSTUS, and TSEBRA. *Genome Research*, page genome;gr.278090.123v1, June 2024. ISSN 1088-9051, 1549-5469. doi: 10.1101/gr.278090.123. URL `http://genome.cshlp.org/lookup/doi/10.1101/gr.278090.123`.
14. Felix Mölder, Kim Philipp Jablonski, Brice Letcher, Michael B. Hall, Christopher H. Tomkins-Tinch, Vanessa Sochat, Jan Forster, Soohyun Lee, Sven O. Twardziok, Alexander Kanitz, Andreas Wilm, Manuel Holtgrewe, Sven Rahmann, Sven Nahnsen, and Johannes Köster. Sustainable data analysis with Snakemake. *F1000Research*, 10:33, April 2021. ISSN 2046-1402. doi: 10.12688/f1000research.29032.2. URL `https://f1000research.com/articles/10-33/v2`.
15. Sarah D. Kocher, Ricardo Mallarino, Benjamin E. R. Rubin, Douglas W. Yu, Hopi E. Hoekstra, and Naomi E. Pierce. The genetic basis of a social polymorphism in halictid bees. *Nature Communications*, 9(1):4338, October 2018. ISSN 2041-1723. doi: 10.1038/s41467-018-06824-8. URL `https://www.nature.com/articles/s41467-018-06824-8`.
16. Scott W. Wolf, Dee M. Ruttenberg, Daniel Y. Knapp, Andrew E. Webb, Ian M. Traniello, Grace C. McKenzie-Smith, Sophie A. Leheny, Joshua W. Shaevitz, and Sarah D. Kocher. NAPS: Integrating pose estimation and tag-based tracking. *Methods in Ecology and Evolution*, 14(10):2541–2548, October 2023. ISSN 2041-210X, 2041-210X. doi: 10.1111/2041-210X.14201. URL `https://besjournals.onlinelibrary.wiley.com/doi/10.1111/2041-210X.14201`.
17. Anaconda Inc. Anaconda software distribution. URL `https://docs.anaconda.com/`.
18. Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. Singularity: Scientific containers for mobility of compute. *PLOS ONE*, 12(5):e0177459, May 2017. ISSN 1932-6203. doi: 10.1371/journal.pone.0177459. URL `https://dx.plos.org/10.1371/journal.pone.0177459`.
19. Docker, Inc. Docker Hub. URL `https://hub.docker.com/`.
20. D. L. Wheeler, T. Barrett, D. A. Benson, S. H. Bryant, K. Canese, V. Chetvernin, D. M. Church, M. DiCuccio, R. Edgar, S. Federhen, M. Feolo, L. Y. Geer, W. Helmberg, Y. Kapustin, O. Khovayko, D. Landsman, D. J. Lipman, T. L. Madden, D. R. Maglott, V. Miller, J. Ostell, K. D. Pruitt, G. D. Schuler, M. Shumway, E. Sequeira, S. T. Sherry, K. Sirotkin, A. Souvorov, G. Starchenko, R. L. Tatusov, T. A. Tatusova, L. Wagner, and E. Yaschenko. Database resources of the National Center for Biotechnology Information. *Nucleic Acids Research*, 36(Database):D13–D21, December 2007. ISSN 0305-1048, 1362-4962. doi: 10.1093/nar/gkm1000. URL `https://academic.oup.com/nar/article-lookup/doi/10.1093/nar/gkm1000`.
21. Daehwan Kim, Joseph M. Paggi, Chanhee Park, Christopher Bennett, and Steven L. Salzberg. Graph-based genome alignment and genotyping with HISAT2 and HISAT-genotype. *Nature Biotechnology*, 37(8):907–915, August 2019. ISSN 1087-0156, 1546-1696. doi: 10.1038/s41587-019-0201-4. URL `https://www.nature.com/articles/s41587-019-0201-4`.
22. Smit, Arian, Hubley, Robert, and Green, Phil. RepeatMasker Open-4.0, 2013. URL `https://www.repeatmasker.org`.
23. Jullien M. Flynn, Robert Hubley, Clément Goubert, Jeb Rosen, Andrew G. Clark, Cédric Feschotte, and Arian F. Smit. RepeatModeler2 for automated genomic discovery of transposable element families. *Proceedings of the National Academy of Sciences*, 117(17):9451–9457, April 2020. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.1921046117. URL `https://pnas.org/doi/full/10.1073/pnas.1921046117`.
24. Beryl M Jones, Andrew E Webb, Scott M Geib, Sheina Sim, Rena M Schweizer, Michael G Branstetter, Jay D Evans, and Sarah D Kocher. Repeated Shifts in Sociality Are Associated With Fine-tuning of Highly Conserved and Lineage-Specific Enhancers in a Socially Flexible Bee. *Molecular Biology and Evolution*, 41(11):msae229, November 2024. ISSN 0737-4038, 1537-1719. doi: 10.1093/molbev/msae229. URL `https://academic.oup.com/mbe/article/doi/10.1093/molbev/msae229/7863458`.
25. Adrian M Altenhoff, Alex Warwick Vesztrocy, Charles Bernard, Clement-Marie Train, Alina Nicheperovich, Silvia Prieto Baños, Irene Julca, David Moi, Yannis Nevers, Sina Majidian, Christophe Dessimoz, and Natasha M Glover. OMA orthology in 2024: improved prokaryote coverage, ancestral and extant GO enrichment, a revamped synteny viewer and more in the OMA Ecosystem. *Nucleic Acids Research*, 52 (D1):D513–D521, January 2024. ISSN 0305-1048, 1362-4962. doi: 10.1093/nar/gkad1020. URL `https://academic.oup.com/nar/article/52/D1/D513/7420097`.
26. Evgeny M Zdobnov, Dmitry Kuznetsov, Fredrik Tegenfeldt, Mosè Manni, Matthew Berkeley, and Evgenia V Kriventseva. OrthoDB in 2020: evolutionary and functional annotations of orthologs. *Nucleic Acids Research*, 49(D1):D389–D393, January 2021. ISSN 0305-1048, 1362-4962. doi: 10.1093/nar/gkaa1009. URL `https://academic.oup.com/nar/article/49/D1/D389/5983625`.
27. Cade D Mirchandani, Allison J Shultz, Gregg W C Thomas, Sara J Smith, Mara Baylis, Brian Arnold, Russ Corbett-Detig, Erik Enbody, and Timothy B Sackton. A Fast, Reproducible, High-throughput Variant Calling Workflow for Population Genomics. *Molecular Biology and Evolution*, 41 (1):msad270, January 2024. ISSN 1537-1719. doi: 10.1093/

molbev/msad270. URL https://doi.org/10.1093/molbev/msad270.

28. Xiang Zhou and Matthew Stephens. Genome-wide efficient mixed-model analysis for association studies. *Nature Genetics*, 44(7):821–824, July 2012. ISSN 1061-4036, 1546-1718. doi: 10.1038/ng.2310. URL https://www.nature.com/articles/ng.2310.
29. Talmo D Pereira, Nathaniel Tabris, Arie Matsliah, David M Turner, Junyu Li, Shruthi Ravindranath, Eleni S Papadoyannis, Edna Normand, David S Deutsch, Z. Yan Wang, Grace C McKenzie-Smith, Catalin C Mitelut, Marielisa Diez Castro, John D'Uva, Mikhail Kislin, Dan H Sanes, Sarah D Kocher, Samuel S-H, Annegret L Falkner, Joshua W Shaevitz, and Mala Murthy. Sleap: A deep learning system for multi-animal pose tracking. *Nature Methods*, 19(4), 2022.
30. Alexander Mathis, Pranav Mamidanna, Kevin M. Cury, Taiga Abe, Venkatesh N. Murthy, Mackenzie W. Mathis, and Matthias Bethge. Deeplabcut: markerless pose estimation of user-defined body parts with deep learning. *Nature Neuroscience*, 2018. URL https://www.nature.com/articles/s41593-018-0209-y.