

REPRODUCIBLE BACTERIAL GENOMICS ON MODEST HARDWARE: AN
AUTOMATION AND OFFLOADING FRAMEWORK FOR
ANTIMICROBIAL RESISTANCE AND PATHOGEN WHOLE GENOME
ANALYSIS, WITH ESBL PRODUCING ESCHERICHIA COLI AS A CASE
STUDY

Saadia Fazal^{*1}, Dr. Sammiya Abrar²

^{*1,2}Department of Biological Sciences, The Superior University, Lahore, Pakistan

^{*1}saadiafazal111@gmail.com

DOI: <https://doi.org/10.5281/zenodo.22277301>

Keywords

bacterial whole genome
sequencing; antimicrobial
resistance; reproducibility;
workflow automation; Galaxy;
pan genome; phylogenomics; low
resource bioinformatics;
Escherichia coli

Article History

Received: 28 June 2026

Accepted: 02 August 2026

Published: 15 August 2026

Copyright @Author

Corresponding Author: *

Saadia Fazal

Abstract

Whole genome sequencing (WGS) has become central to bacterial epidemiology and antimicrobial resistance (AMR) research, but the analytical workflow that turns raw genomes into interpretable results is fragmented across many command line tools and is frequently run by hand. Manual execution is slow. It is hard to reproduce and error prone. The problem is sharpest for the many laboratories particularly in low and middle income settings that work on modest desktop hardware rather than high performance computing (HPC) clusters. Here we present a practical framework for converting fragmented bacterial WGS analyses into automated, reproducible workflows that remain feasible on a standard laptop. We first map the canonical comparative genomics pipeline. This pipeline will help in genome retrieval, quality control, annotation, resistome, virulome and plasmidome profiling, sequence typing, pan genome reconstruction, phylogenetics and integration. It will identify the recurring bottlenecks at each stage. We then describe a lightweight automation pattern built from widely available Linux and bioinformatics tools. A single machine readable manifest as the source of truth, configuration driven and modular scripts, pinned software environments, fixed random seeds and structured logging as an audit trail. The framework's distinctive element is an explicit local versus offload decision rule. The memory and CPU light steps are automated locally. While a small number of resource intensive steps such as core genome phylogenetics offloaded to a free web platform (Galaxy) or the cloud. We illustrate the framework with a completed comparative genomic study of 100 ESBL producing E. coli genomes performed end to end on an 8 GB laptop. Every step ran locally except the maximum likelihood phylogeny, which exceeded available memory and was offloaded successfully. We provide a decision table and reproducibility checklist that researchers can adopt directly. The framework lowers the hardware barrier to reproducible pathogen genomics without requiring migration to a full workflow management system, while remaining fully compatible with such systems as projects scale.

1. INTRODUCTION

Bacterial whole genome sequencing (WGS) has moved from a specialist technique to a routine instrument of infection surveillance, outbreak investigation and antimicrobial resistance (AMR) research. A single assembled genome can now be interrogated for acquired resistance genes and resistance conferring mutations, virulence determinants, plasmid replicons, sequence type and phylogenetic position. The comparative studies routinely analyse tens to thousands of genomes at once. Yet the analytical layer that converts genomes into biological conclusions has not kept pace with the sequencing itself. It remains a chain of independently developed command line tools. Each tool with its own input and output formats, dependencies and idioms, and in many laboratories that chain is still executed largely by hand.

Manual execution carries three well recognised costs. It is slow, because a human must shepherd each step and wait on the previous one. It is error prone, because parameters, file paths and intermediate versions are easy to confuse across dozens of commands. It is difficult to reproduce, because the exact sequence of operations, tool versions and reference databases is rarely captured in a form that another researcher or the same researcher months later can re run. The reproducibility of computational biology has been questioned repeatedly. Bacterial genomics is not exempt from this claim. Results depend not only on the algorithms but on the specific database releases, parameter choices and even file format conventions used on the day.

These problems bite hardest in laboratories without access to high performance computing (HPC) clusters or cloud budgets. A large fraction of pathogen genomics research now originates in low and middle income countries. These are the places where the burden of drug resistant infection is highest. The available hardware is often a single desktop or laptop with limited memory in these regions. Much of the existing methodological literature implicitly assumes an HPC environment, and the guidance on how to make a WGS analysis both automated and feasible on

modest hardware is correspondingly thin. The practical question which steps can be automated locally, which cannot, and what to do about the ones that cannot has largely been left to individual trial and error.

This review addresses that question directly. We ask: **how can commonly used Linux based and bioinformatics tools be integrated into automated, reproducible workflows that reduce manual effort, analysis time and error in bacterial WGS research, while remaining feasible on modest hardware?** We do not aim to catalogue tools. Many excellent tool reviews exist but to present a structured framework for converting fragmented analyses into automated, reproducible pipelines, with an explicit rule for deciding when to compute locally and when to offload. We ground the framework in a completed comparative genomic study of 100 ESBL producing *E. coli* genomes carried out end to end on an 8 GB laptop. Our work serves throughout as a concrete, honest case study including the point at which local computation failed and had to be offloaded.

The contribution is therefore threefold: (i) a stage by stage account of the bacterial comparative genomics pipeline and its recurring bottlenecks, (ii) a lightweight automation and reproducibility pattern assembled from freely available tools. It neither require an HPC cluster nor a full workflow management system and (iii) an explicit local versus offload decision framework, distilled into a decision table and a reproducibility checklist that can be adopted directly. The intended readership is the working pathogen genomics researcher especially the student or resource limited laboratory for whom the barrier to reproducible analysis is as much practical as conceptual.

2. The manual bacterial WGS workflow and where it breaks down

The comparative genomics workflow for bacterial isolates is conceptually linear but practically fragmented. We describe it here as the sequence of stages a researcher actually performs, and at each stage name the bottleneck that automation must address (summarised in Fig. 1 and Table 1).

2.1. Genome retrieval and manifest construction

Studies that reuse public data begin by selecting genomes from repositories such as NCBI GenBank or ENA, often from hundreds of thousands of candidate assemblies. Done manually, this means browsing web interfaces, copying accessions and downloading files one by one. **Bottleneck:** selection is slow, undocumented and effectively irreproducible. Another researcher cannot recover which of 400,000 assemblies were chosen or why. The automation target is a single machine readable *manifest* that records every included accession together with the eligibility rules, ranking criteria and random seed used to build it, so that the exact dataset can be regenerated.

2.2. Quality control

Each assembly must be screened for completeness, contamination, structural integrity and correct species identity before it enters downstream analysis. Multiple tools are involved. Assembly statistics, average nucleotide identity checks and completeness/contamination estimation each producing its own report. **Bottleneck:** manually collating heterogeneous QC reports across many genomes and applying consistent pass/fail thresholds is tedious and easy to get subtly wrong. The automation target is a single pass/fail rule applied uniformly and recorded per genome.

2.3. Annotation

Public assemblies arrive with heterogeneous, non comparable annotations produced by different pipelines and database versions. Reliable comparative analysis especially pan genome reconstruction requires that every genome be re annotated with a single tool and database. **Bottleneck:** annotation is computationally heavier than QC and must be applied identically to every genome. If done by hand across a collection it is slow and inconsistent. The automation target is uniform re annotation driven from the manifest.

2.4. Resistome, virulome and plasmidome profiling

Resistance determinants, virulence factors and plasmid replicons are identified by screening each genome against curated reference databases. Each tool emits a per genome table in its own format. **Bottleneck:** the scientific object of interest is not the individual tables but a combined gene by genome presence/absence matrix across the whole collection. Assembling that matrix by hand from dozens of tab separated files is a classic source of transcription error. The automation target is programmatic parsing and aggregation into a single matrix per domain.

2.5. Sequence typing and pan genome reconstruction

Multi locus sequence typing assigns each genome to a lineage. Pan genome reconstruction partitions the collection's genes into core and accessory compartments and produces a core genome alignment. **Bottleneck:** pan genome tools are sensitive to annotation format and consume substantial memory as genome number grows, format mismatches silently exclude inputs, and large collections can exceed desktop memory. The automation target is format normalisation before pan genome reconstruction and a clear memory budget for the step.

2.6. Phylogenetics

A maximum likelihood phylogeny from the core genome alignment places the isolates in evolutionary context. This is typically the single most resource intensive step. Model selection and bootstrapping over a multi thousand gene alignment are memory and CPU hungry. **Bottleneck:** on modest hardware this step frequently exceeds available memory and either fails outright or runs for many hours without completing. It is the canonical case where local automation alone is insufficient and offloading must be considered.

2.7. Integration, statistics and interpretation

Finally, the per domain outputs are merged into a single per genome table for statistical comparison

and increasingly machine learning analysis. **Bottleneck:** if each preceding step was run by hand, the integration inherits every inconsistency upstream. Keys fail to match, versions diverge and the final table is hard to trust. The automation target is a single keyed master table assembled programmatically from validated intermediate outputs.

Two properties of this pipeline motivate the framework that follows. First, the bottlenecks are overwhelmingly about *orchestration and format* that are downloads, collation, parsing, key matching, datatype tags rather than about the algorithms themselves. Reproducibility failures in practice are usually format or metadata failures, not algorithmic ones. Second, the computational cost is highly uneven. Most steps are light enough to automate on a laptop. A small number chiefly phylogenetics and, for large collections, pan genome reconstruction are not. An effective framework must therefore address both automation and the local versus offload decision.

3. A lightweight automation and reproducibility framework

The goal is not to adopt the heaviest possible infrastructure but the lightest one that makes an analysis automated, reproducible and portable. Five elements, all built from freely available tools, achieve this without requiring a workflow management system or a cluster.

3.1. A manifest as the single source of truth

The foundation is one machine readable table a manifest. This manifest will list every genome together with its accession, provenance and the eligibility and ranking rules used to select it. Every downstream step reads its inputs from the manifest and writes its status back to it. By doing so the dataset is defined in exactly one place. Building the manifest by streaming the source records one at a time, rather than loading them all into memory. This keeps memory use flat regardless of how large the candidate pool is an important property when the pool numbers in the hundreds of thousands and the machine has only a few gigabytes of RAM. The manifest is what

makes selection reproducible. Given the same rules and seed, the same genomes are recovered.

3.2. Configuration driven, modular scripts

Each stage of the pipeline is a small, single purpose script that takes its parameters from a central configuration file rather than from hard coded values. Paths, thresholds, database versions and the random seed live in one place, so the same code runs on a different machine or dataset by editing configuration alone. Numbering the scripts in execution order (retrieval, QC, annotation, and so on) turns the pipeline into a self documenting sequence that a newcomer can follow and re run. This pattern captures most of the reproducibility benefit of a formal workflow manager at a fraction of the setup cost, and it scales down gracefully to a single laptop.

3.3. Pinned software environments

Because results depend on tool and database versions, those versions must be fixed and recorded. Lightweight environment managers such as conda/mamba let each project declare its exact dependencies in a file that can be recreated elsewhere for full portability. The same environment can be built into a container (Docker or, on shared systems that disallow root, Singularity/Apptainer [22], community images via BioContainers [23]). Recording the version of every tool and every reference database used resistance catalogue, virulence database, plasmid database, typing scheme is not a formality but a precondition for another laboratory to obtain the same calls.

3.4. Deterministic analysis: fixed seeds and uniform rules

Any step involving randomness genome subsampling, cross validation splits, tie breaking during selection must use a fixed seed so that the analysis is deterministic. Similarly, applying one pass/fail QC rule and one statistical procedure uniformly across the whole collection, rather than case by case, removes a large class of silent inconsistencies. Determinism is what allows a

reviewer to reproduce a figure exactly rather than approximately.

3.5. Structured logging as an audit trail

Every step should write a dated log recording the exact command run, the tool version and the outcome. Beyond aiding debugging, these logs constitute an audit trail. The trail depends on that documents precisely how each result was produced. When a long step is launched as a background process, its log is often the only record that it ran and completed. Systems where a closed terminal can terminate detached jobs, that record is also the first place to check after an unexpected failure.

4. The local versus offload decision

Automation makes a pipeline reproducible. It does not make an over large computation fit in memory. The framework's one non obvious claim is this. On modest hardware, decide in advance which steps run locally and which get offloaded. Instead of hitting the memory ceiling halfway through an analysis and improvising. The decision turns on a small number of factors.

Memory footprint. The dominant constraint on a laptop is RAM. Retrieval, QC, annotation, resistome/virulome/plasmidome screening, typing and the integration/statistics layer are all comfortably within a few gigabytes for collections of dozens to low hundreds of genomes. Pan genome reconstruction grows with genome number. It can approach the ceiling for large collections. Maximum likelihood phylogenetics over a multi thousand gene core alignment is the step most likely to exceed a small memory budget outright.

Runtime and interactivity. Steps that take minutes belong locally. Steps that take many hours, or that must be restarted repeatedly to tune, are candidates for offloading to a platform where they run server side and survive local interruptions.

Reproducibility of the remote step. Offloading is acceptable only if the remote computation is itself reproducible. Public platforms such as Galaxy [5] record tool versions and parameters and expose

the exact settings used, so an offloaded step remains documented. ad hoc remote execution without version capture does not qualify.

Applying these factors yields a simple rule (Table 2). Automate locally everything that fits comfortably in memory and completes quickly. Offload the few steps that do not principally phylogenetics, and pan genome reconstruction for large collections to a free web platform such as Galaxy [5] or, where available, to cloud compute. Offloading is not a failure of automation but a component of it. The pipeline is designed from the outset with a defined boundary between local and remote steps. The manifest and intermediate files are the interface across that boundary. Free platforms are emphasised deliberately. The laboratories most constrained by local hardware are also those least likely to hold cloud budgets. Galaxy in particular provides substantial server side memory at no cost. It records the provenance needed for reproducibility as well.

5. Reproducibility and portability in practice

Two recurring, practical issues deserve emphasis because they cause a disproportionate share of lost time and irreproducible results, and neither is algorithmic.

Format and datatype mismatches. Tools frequently reject inputs not because the data are wrong but because a format tag does not match what the tool expects. A representative example arises when supplying annotations to a pan genome tool. If the annotation files are tagged as generic GFF rather than GFF3, the tool silently omits them from its input list. The analysis will appear to have no data. The fix is simply to assign the correct datatype. An operation that takes seconds once the cause is understood, but that can cost an afternoon when it is not. The general lesson is that reproducibility hinges on metadata and format conventions as much as on the analysis itself. The pipelines should normalise formats explicitly before each sensitive step.

Version control and archiving. Scripts, the configuration file and the environment specification should be tracked in version control (Git) and archived at publication, ideally with a

citable identifier. Raw genomes need not be re deposited when they originate from a public repository. Recording their accessions in the manifest is sufficient and preferable, since it points to the authoritative source. Derived outputs the presence/absence matrices, the master table, the tree can be released as supplementary data so that others can reproduce the figures without re running the whole pipeline.

Adopting these practices does not require migrating to a formal workflow management system. Systems such as Snakemake [2,3] and Nextflow [4] (and curated collections such as nf core) are powerful and become worthwhile as pipelines are re run at scale or shared across teams. The framework described here is fully compatible with them [1]. The manifest, configuration and pinned environments transfer directly. The point is that the reproducibility benefits are available immediately, at laptop scale, without that migration. A researcher can adopt a workflow manager later without discarding any of the earlier structure.

6. Case study: comparative genomics of ESBL producing *E. coli* on an 8 GB laptop

We illustrate the framework with a completed study that applied it end to end to 100 ESBL producing *E. coli* genomes. 50 from Pakistan and 50 from ten developed countries. Selected from an initial pool of 411,460 GenBank assemblies. The entire analysis was performed on a single laptop with 8 GB of RAM running Ubuntu 24.04 under the Windows Subsystem for Linux (WSL2), with tools installed in conda/mamba environments (Python 3.12).

Every element of the framework was used. A streamed manifest recorded the eligibility rules (which reduced 411,460 assemblies to 1,099 eligible). The ranking that preferred clinically sourced, higher quality assemblies, and the fixed seed used for tie breaking, so that the final balanced set of 100 genomes is exactly reproducible. Quality control (QUAST [7], skani [8], CheckM2 [9]) applied a single pass/fail rule, retaining 99 of 100 genomes and triggering one documented replacement. Uniform re annotation

(Bakta [10]) produced comparable gene calls across the collection. Resistome (AMRFinderPlus [11]), virulome (VFDB [12] via ABRicate) and plasmidome (PlasmidFinder [13]) screening were parsed programmatically into presence/absence matrices, sequence typing (mlst; PubMLST [14]) and pan genome reconstruction (Panaroo [15]) followed. All of these steps ran locally within the 8 GB budget.

The offload point. The single step that could not be completed locally was the maximum likelihood phylogeny. Model selection and bootstrapping over the multi thousand gene core alignment repeatedly exceeded available memory. By increasing swap space allowed the process to grind for many hours without producing a tree, and abrupt termination of the environment could kill the detached job entirely. In line with the decision rule, this step and only this step was offloaded to the free Galaxy platform [5]. IQ TREE [16] ran server side on high memory infrastructure and completed successfully. Recording the exact model and parameters (GTR+F+R8, selected by ModelFinder [17]. 1,000 ultrafast bootstrap replicates [18]) for reproducibility. The resulting tree, together with the sequence types, produced the study's central result. The rest of the pipeline, including the integration layer and a machine learning cross check (scikit learn [20]. Statistics with SciPy [19]), returned to the laptop.

The case study demonstrates the framework's core proposition concretely. A genuine, publishable comparative genomics analysis of 100 genomes is achievable on commodity hardware. Only when the pipeline is automated, reproducible, and designed from the outset with an explicit boundary between the steps that run locally and the one step that must be offloaded. Full biological findings of that study are reported separately. Here it serves only to validate the framework. We report the analysis qualitatively rather than as a formal benchmark. The point is not a timing comparison but the structural division of labour between the many steps that ran within the laptop's memory and the single step that did not.

7. Recommended implementation and decision aids

Table 1 maps each pipeline stage to its bottleneck and automation target. Table 2 gives the local versus offload decision rule and the checklist below distills the framework into concrete actions a researcher can adopt immediately.

Reproducibility checklist.

- Define the dataset in a single manifest recording accessions, eligibility rules, ranking and random seed.
- Drive every step from a central configuration file. Number scripts in execution order.
- Pin tool and reference database versions in an environment file.
- Fix all random seeds. Apply one QC rule and one statistical procedure uniformly across the collection.
- Write a dated log for every step recording the command, tool version and outcome.
- Normalise file formats/datatypes explicitly before format sensitive steps (e.g. GFF3 for pan genome tools).
- Decide the local/offload boundary in advance from the memory budget; offload only to platforms that record versions and parameters.
- Track scripts, configuration and environment in version control; archive them at

publication with a citable identifier. Cite genome accessions rather than re depositing public data.

8. Conclusion

Bacterial WGS analysis does not have to be either manual or confined to well funded laboratories with HPC access. The bottlenecks that make manual pipelines slow and irreproducible are predominantly problems of orchestration and format. They yield to a lightweight automation pattern. A manifest, configuration driven modular scripts, pinned environments, fixed seeds and structured logging that can be assembled entirely from freely available tools and run on a laptop. The one genuinely hardware bound step, core genome phylogenetics, is handled not by abandoning modest hardware but by planning an explicit offload to a free, reproducible web platform. The result is a framework that lowers the barrier to reproducible pathogen genomics for the researchers who most need it. While remaining a natural on ramp to formal workflow management systems as projects grow. We have shown, through a completed 100 genome ESBL producing *E. coli* study performed on an 8 GB laptop, that the approach is not merely aspirational but sufficient for real, publishable comparative genomics.

Tables

Table 1. Stages of the bacterial comparative genomics pipeline, the principal bottleneck of manual execution at each stage, and the automation target.

Stage	Manual bottleneck	Automation target
Genome retrieval	Undocumented, one by one selection from very large pools	Streamed manifest recording accessions, rules and seed
Quality control	Collating heterogeneous QC reports and inconsistent thresholds	One pass/fail rule applied and recorded uniformly
Annotation	Heterogeneous public annotations. Slow manual re runs	Uniform re annotation driven from the manifest
Resistome/virulome/plasmidome	Hand assembling matrices from many per genome tables	Programmatic parsing into presence vs absence matrices
Typing & pan genome	Format sensitivity. Memory growth with genome number	Format normalization, defined memory budget

Stage	Manual bottleneck	Automation target
Phylogenetics	Exceeds laptop memory. Long or failed runs	Planned offload to high memory platform
Integration & statistics	Inherits upstream inconsistencies, key mismatches	Single keyed master table from validated outputs

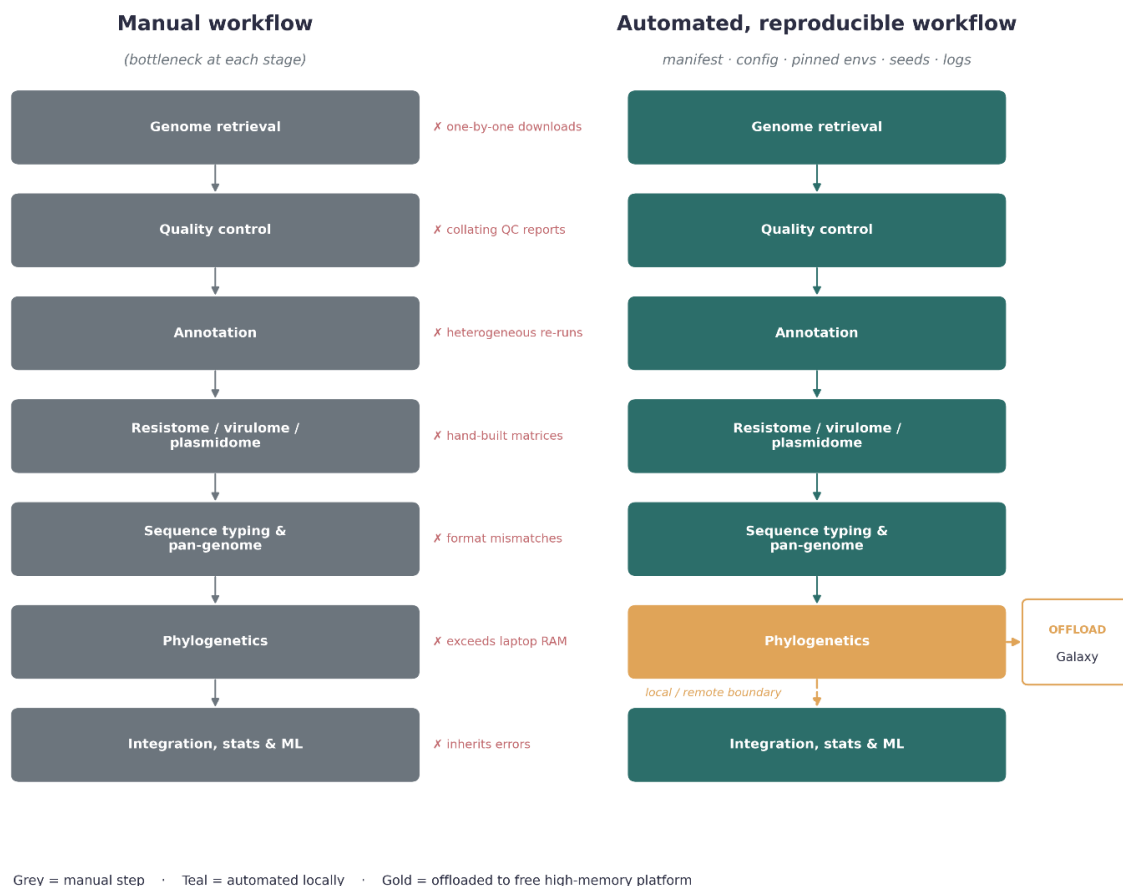
Table 2. Local versus offload decision rule for a modest hardware (e.g. 8 GB RAM) environment. “Offload” denotes a free, provenance recording platform such as Galaxy, or cloud compute where available.

Step	Typical memory demand	Recommended venue
Retrieval, QC, annotation	Low	Local
Resistome / virulome / plasmidome	Low	Local
Sequence typing	Low	Local
Pan genome reconstruction	Moderate but grows with genome count	Local for small/medium. Offload for large collections
Core genome ML phylogeny	High	Offload (canonical case)
Integration, statistics, ML	Low	Local

Figure legend

Fig. 1. The bacterial comparative genomics pipeline as a manual workflow (left) and as the proposed automated, reproducible workflow with

an explicit local/offload boundary (right). Each stage is annotated with its manual bottleneck. The phylogenetics stage crosses the boundary to a free high memory platform (Galaxy).



Declarations

Data and code availability. No new genome sequences were generated. The 100 genomes analysed in the case study are publicly available NCBI assemblies, listed by accession in the associated primary study. The derived outputs of the case study: the resistome, virulome and plasmidome presence/absence matrices, the integrated master table and the core genome phylogeny along with the analysis code are available at [repository URL]. [6].

Funding. No specific funding.

Competing interests. The author declares none.

Author contributions (CRediT). Saadia Fazal: Conceptualization, Writing original draft, Writing review & editing, Visualization.

REFERENCES

- [1] Wratten L, Wilm A, Göke J. Reproducible, scalable, and shareable analysis pipelines with bioinformatics workflow managers. *Nat Methods*. 2021;18(10):1161-8. doi:10.1038/s41592-021-01254-9.
- [2] Köster J, Rahmann S. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*. 2012;28(19):2520-2. doi:10.1093/bioinformatics/bts480.
- [3] Mölder F, Jablonski KP, Letcher B, Hall MB, Tomkins Tinch CH, Sochat V, et al. Sustainable data analysis with Snakemake. *F1000Research*. 2021;10:33. doi:10.12688/f1000research.29032.2.

- [4] Di Tommaso P, Chatzou M, Floden EW, Barja PP, Palumbo E, Notredame C. Nextflow enables reproducible computational workflows. *Nat Biotechnol.* 2017;35(4):316–9. doi:10.1038/nbt.3820.
- [5] The Galaxy Community. The Galaxy platform for accessible, reproducible, and collaborative data analyses: 2024 update. *Nucleic Acids Res.* 2024;52(W1):W83–W94. doi:10.1093/nar/gkae410.
- [6] Sayers EW, Bolton EE, Brister JR, Canese K, Chan J, Comeau DC, et al. Database resources of the National Center for Biotechnology Information. *Nucleic Acids Res.* 2022;50(D1):D20–D26. doi:10.1093/nar/gkab1112.
- [7] Gurevich A, Saveliev V, Vyahhi N, Tesler G. QUAST: quality assessment tool for genome assemblies. *Bioinformatics.* 2013;29(8):1072–5. doi:10.1093/bioinformatics/btt086.
- [8] Shaw J, Yu YW. Fast and robust metagenomic sequence comparison through sparse chaining with skani. *Nat Methods.* 2023;20(11):1661–5. doi:10.1038/s41592-023-02018-3.
- [9] Chklovski A, Parks DH, Woodcroft BJ, Tyson GW. CheckM2: a rapid, scalable and accurate tool for assessing microbial genome quality using machine learning. *Nat Methods.* 2023;20(8):1203–12. doi:10.1038/s41592-023-01940-w.
- [10] Schwengers O, Jelonek L, Dieckmann MA, Beyvers S, Blom J, Goesmann A. Bakta: rapid and standardized annotation of bacterial genomes via alignment free sequence identification. *Microb Genom.* 2021;7(11):000685. doi:10.1099/mgen.0.000685.
- [11] Feldgarden M, Brover V, Gonzalez Escalona N, Frye JG, Haendiges J, Haft DH, et al. AMRFinderPlus and the Reference Gene Catalog facilitate examination of the genomic links among antimicrobial resistance, stress response, and virulence. *Sci Rep.* 2021;11(1):12728. doi:10.1038/s41598-021-91456-0.
- [12] Liu B, Zheng D, Zhou S, Chen L, Yang J. VFDB 2022: a general classification scheme for bacterial virulence factors. *Nucleic Acids Res.* 2022;50(D1):D912–7. doi:10.1093/nar/gkab1107.
- [13] Carattoli A, Zankari E, García Fernández A, Voldby Larsen M, Lund O, Villa L, et al. In silico detection and typing of plasmids using PlasmidFinder and plasmid multilocus sequence typing. *Antimicrob Agents Chemother.* 2014;58(7):3895–903. doi:10.1128/AAC.02412-14.
- [14] Jolley KA, Bray JE, Maiden MCJ. Open access bacterial population genomics: BIGSdb software, the PubMLST.org website and their applications. *Wellcome Open Res.* 2018;3:124. doi:10.12688/wellcomeopenres.14826.1.
- [15] Tonkin Hill G, MacAlasdair N, Ruis C, Weimann A, Horesh G, Lees JA, et al. Producing polished prokaryotic pangenomes with the Panaroo pipeline. *Genome Biol.* 2020;21(1):180. doi:10.1186/s13059-020-02090-4.
- [16] Minh BQ, Schmidt HA, Chernomor O, Schrempf D, Woodhams MD, von Haeseler A, Lanfear R. IQ TREE 2: new models and efficient methods for phylogenetic inference in the genomic era. *Mol Biol Evol.* 2020;37(5):1530–4. doi:10.1093/molbev/msaa015.
- [17] Kalyaanamoorthy S, Minh BQ, Wong TKF, von Haeseler A, Jermini LS. ModelFinder: fast model selection for accurate phylogenetic estimates. *Nat Methods.* 2017;14(6):587–9. doi:10.1038/nmeth.4285.
- [18] Hoang DT, Chernomor O, von Haeseler A, Minh BQ, Vinh LS. UFBoot2: improving the ultrafast bootstrap approximation. *Mol Biol Evol.* 2018;35(2):518–22. doi:10.1093/molbev/msx281.

- [19] Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nat Methods*. 2020;17(3):261–72. doi:10.1038/s41592-019-0686-2.
- [20] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit learn: machine learning in Python. *J Mach Learn Res*. 2011;12:2825–30.
- [21] Grüning B, Dale R, Sjödin A, Chapman BA, Rowe J, Tomkins Tinch CH, et al. Bioconda: sustainable and comprehensive software distribution for the life sciences. *Nat Methods*. 2018;15(7):475–6. doi:10.1038/s41592-018-0046-7.
- [22] Kurtzer GM, Sochat V, Bauer MW. Singularity: scientific containers for mobility of compute. *PLoS One*. 2017;12(5):e0177459. doi:10.1371/journal.pone.0177459.
- [23] da Veiga Leprevost F, Grüning BA, Alves Aflitos S, Röst HL, Uszkoreit J, Barsnes H, et al. BioContainers: an open source and community driven framework for software standardization. *Bioinformatics*. 2017;33(16):2580–2. doi:10.1093/bioinformatics/btx192.