The Journal of Open Source Software

Viash: A meta-framework for building reusable workflow modules

Robrecht Cannoodt [1,2,3], **Hendrik Cannoodt** [1], **Dries Schaumont** [1], **Kai Waldrant** [1], **Eric Van de Kerckhove**[1], **Andy Boschmans** [1], **Dries De Maeyer** [4], **and Toni Verbeiren** [1]

1 Data Intuitive, Lebbeke, Belgium **2** Data Mining and Modelling for Biomedicine group, VIB Center for Inflammation Research, Ghent, Belgium **3** Department of Applied Mathematics, Computer Science, and Statistics, Ghent University, Ghent, Belgium **4** Discovery Technology and Molecular Pharmacology, Janssen Research & Development, Pharmaceutical Companies of Johnson & Johnson, Beerse, Belgium

DOI: 10.21105/joss.06089

Software

- Review
- Repository
- Archive

Editor: Daniel S. Katz

Reviewers:

- @wilkinson
- @abhishektiwari
- @mberacochea

Submitted: 21 November 2023
Published: 24 January 2024

Abstract

Most bioinformatics workflows consist of software components that are tightly coupled to the logic of the workflow itself. This limits reusability of the individual components in the workflow or introduces maintenance overhead when they need to be reimplemented in multiple workflows. We introduce Viash, a tool for speeding up development of robust workflows through "code-first" prototyping, separation of concerns and code generation of modular workflow components. By decoupling the component functionality from the workflow logic, component functionality becomes fully workflow-agnostic, and conversely the resulting workflows are agnostic towards specific component requirements. This separation of concerns improves reusability of components and facilitates multidisciplinary and pan-organisational collaborations. It has been applied in a variety of projects, from proof-of-concept workflows to supporting an international data science competition.

Viash is available as an open-source project at github.com/viash-io/viash and documentation is available at viash.io.

Statement of Need

Recent developments in high-throughput RNA sequencing and imaging technologies allow present-day biologists to observe single-cell characteristics in ever more detail (Luecken & Theis, 2019). As the dataset size and the complexity of bioinformatics workflows increases, so does the need for scalable and reproducible data science. In single cell biology, recent efforts to standardise some of the most common single-cell analyses (Amezquita et al., 2019; Ewels et al., 2020; Heumos et al., 2023) tackle these challenges by using a workflow framework (e.g., Snakemake, Nextflow), containerisation (e.g., Docker, Podman), and horizontal scaling in cloud computing (e.g., Kubernetes, HPC).

Since research projects are increasingly more complex and interdisciplinary, researchers from different fields and backgrounds are required to join forces. This implies that not all project contributors can be experts in computer science. The chosen framework for such projects therefore needs to have a low barrier to entry in order for contributors to be able to participate. One common pitfall that greatly increases the barrier to entry is tightly coupling a workflow and the components it consists of. Major drawbacks include lower transparency of the overall workflow, limited reusability of workflow components, increased complexity, increased debugging time, and a greater amount of time spent refactoring and maintaining boilerplate code. Non-expert developers in particular will experience more arduous debugging sessions as they need to treat the workflow as a black box.

In this work we introduce Viash, a tool for speeding up workflow prototyping through code generation, component modularity, and separation of concerns. With Viash, a user can create a workflow module by writing a small script or using a pre-existing code block, add a small amount of metadata, and use Viash to generate the boilerplate code needed to turn it into a modular Nextflow component. This separates the component functionality from the workflow logic, thereby allowing a component developer to focus on implementing the required functionality using the domain-specific toolkit at hand while being completely agnostic to the chosen workflow framework. Similarly, a workflow developer can design a workflow by chaining together Viash modules while being completely agnostic to the scripting language used in the component.

Core features and functionality

Viash is an open-source embodiment of a ‘code-first' concept for workflow development. Many bioinformatics research projects (and other software development projects) start with prototyping functionality in small scripts or notebooks in order to then migrate the functionality to software packages or workflow frameworks. By adding some metadata to a code block or script (Figure 1A), Viash can turn a (small) code block into a highly malleable object. By encapsulating core functionality in modular building blocks, one can use a Viash component in a myriad of ways (Figure 1B-C): export it as a standalone command-line tool; create a highly intuitive and modular Nextflow component; ensure reproducibility by building, pulling, or pushing Docker containers; or run one or more unit tests to verify that the component works as expected. Integration with CI tools such as GitHub Actions, Jenkins, or Travis CI allows for automation of unit testing, rolling releases, and versioned releases.

The definition of a Viash component – a config and a code block – can be implemented quite concisely (Figure 2 left). Viash currently supports different scripting languages, including Bash, JavaScript, Python, and R. Through the use of several subcommands (Figure 2 right), Viash can build the component into a standalone script using one of three backend platforms – native, Docker, or Nextflow. Additional commands allow processing one or more Viash components simultaneously, e.g., for executing a unit test suite or (re-)building component-specific Docker images.

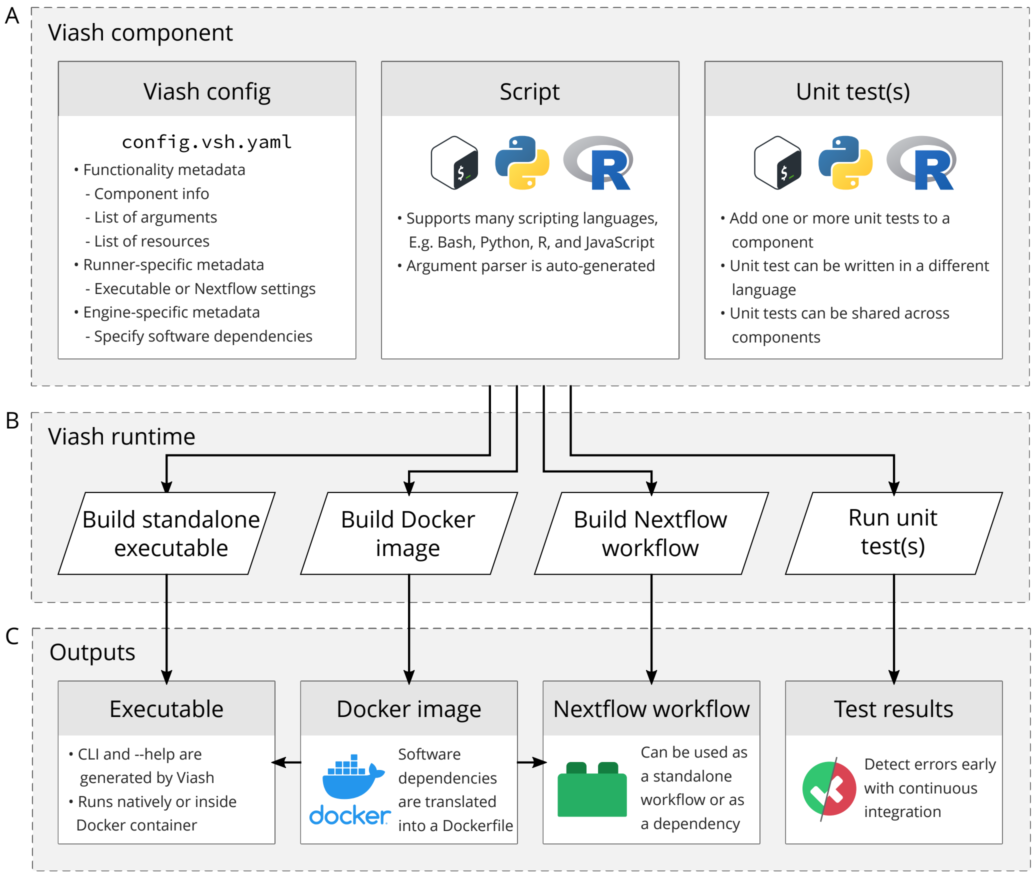
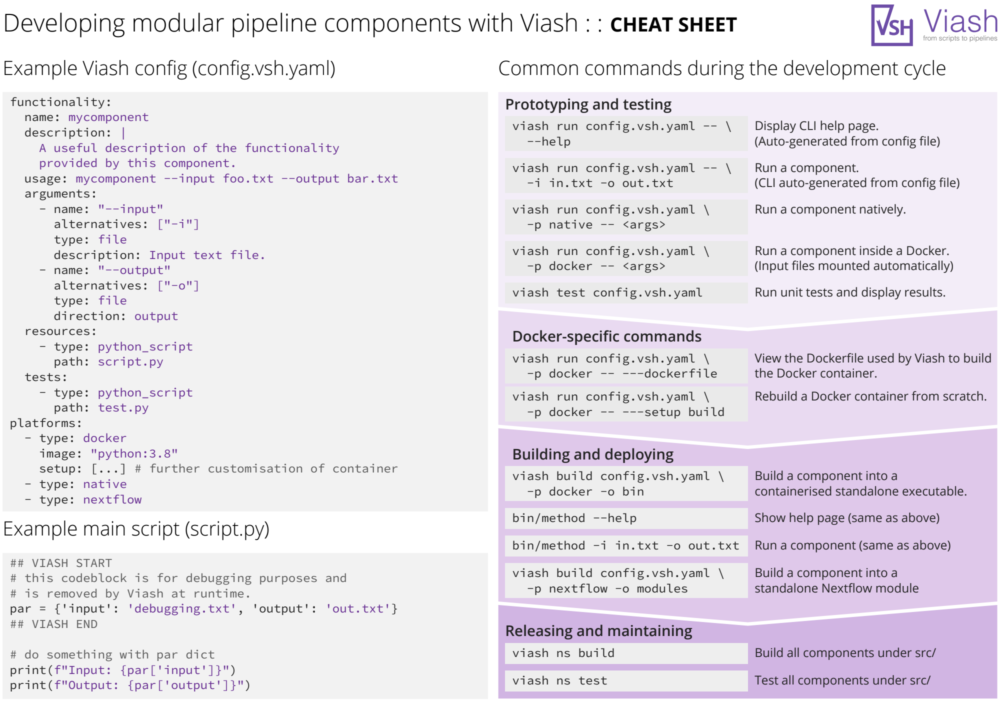

A
Viash component
Viash config
config.vsh.yaml
• Functionality metadata
- Component info
- List of arguments
- List of resources
• Runner-specific metadata
- Executable or Nextflow settings
• Engine-specific metadata
- Specify software dependencies
Script
• Supports many scripting languages, E.g. Bash, Python, R, and JavaScript
• Argument parser is auto-generated
Unit test(s)
• Add one or more unit tests to a component
• Unit test can be written in a different language
• Unit tests can be shared across components
B
Viash runtime
Build standalone executable
Build Docker image
Build Nextflow workflow
Run unit test(s)
C
Outputs
Executable
• CLI and --help are generated by Viash
• Runs natively or inside Docker container
Docker image
docker
Software dependencies are translated into a Dockerfile
Nextflow workflow
Can be used as a standalone workflow or as a dependency
Test results
Detect errors early with continuous integration

Figure 1: Viash allows easy prototyping of reusable workflow components. **A:** Viash requires two main inputs: a script (or code block) and a Viash config file. A Viash config file is a YAML file with metadata describing the functionality provided by the component (e.g., a name and description of the component and its parameters), and platform-specific metadata (e.g., the base Docker container to use, which software packages are required by the component). Optionally, the quality of the component can be improved by defining one or more unit tests with which the component functionality can be tested. **B:** Viash allows transforming a given config to a variety of different outputs. **C:** Viash supports robust workflow development by allowing users to build their component as a standalone executable (with auto-generated CLI), build a Docker image to run the script inside, or turn the component into a standalone Nextflow module or workflow. If unit tests were defined, Viash can also run all of the unit tests and provide users with a report.

One major benefit of using code regeneration is that best practices in workflow development can automatically be applied, whereas otherwise this would be left up to the developer to develop and maintain. For instance, all standalone executables, Nextflow modules, and Docker images are automatically versioned. When parsing command-line arguments, checking for the availability of required parameters, the existence of required input files, or the type-checking of command-line arguments is also automated. Another example is helper functions for installing software through tools such as apt, apk, yum, pip, or R devtools, as these sometimes require additional pre-install commands to update package registries or post-install commands to clean up the installation cache to reduce image size of the resulting image. Here, Viash could be the technical basis for a community of people committed to sharing components that everybody can benefit from.



Developing modular pipeline components with Viash : : **CHEAT SHEET**

Example Viash config (config.vsh.yaml)

```yaml
functionality:
  name: mycomponent
  description: |
    A useful description of the functionality
    provided by this component.
  usage: mycomponent --input foo.txt --output bar.txt
  arguments:
    - name: "--input"
      alternatives: ["-i"]
      type: file
      description: Input text file.
    - name: "--output"
      alternatives: ["-o"]
      type: file
      direction: output
  resources:
    - type: python_script
      path: script.py
  tests:
    - type: python_script
      path: test.py
platforms:
  - type: docker
    image: "python:3.8"
    setup: [...] # further customisation of container
  - type: native
  - type: nextflow
```

Example main script (script.py)

```python
## VIASH START
# this codeblock is for debugging purposes and
# is removed by Viash at runtime.
par = {'input': 'debugging.txt', 'output': 'out.txt'}
## VIASH END

# do something with par dict
print(f"Input: {par['input']}")
print(f"Output: {par['output']}")
```

Common commands during the development cycle

Prototyping and testing

`viash run config.vsh.yaml -- \`
`--help`
	Display CLI help page.
	(Auto-generated from config file)

`viash run config.vsh.yaml -- \`
`-i in.txt -o out.txt`
	Run a component.
	(CLI auto-generated from config file)

`viash run config.vsh.yaml \`
`-p native -- <args>`
	Run a component natively.

`viash run config.vsh.yaml \`
`-p docker -- <args>`
	Run a component inside a Docker.
	(Input files mounted automatically)

`viash test config.vsh.yaml`	Run unit tests and display results.

Docker-specific commands

`viash run config.vsh.yaml \`
`-p docker -- ---dockerfile`
	View the Dockerfile used by Viash to build the Docker container.

`viash run config.vsh.yaml \`
`-p docker -- ---setup build`
	Rebuild a Docker container from scratch.

Building and deploying

`viash build config.vsh.yaml \`
`-p docker -o bin`
	Build a component into a containerised standalone executable.

`bin/method --help`	Show help page (same as above)
`bin/method -i in.txt -o out.txt`	Run a component (same as above)

`viash build config.vsh.yaml \`
`-p nextflow -o modules`
	Build a component into a standalone Nextflow module

Releasing and maintaining

`viash ns build`	Build all components under src/
`viash ns test`	Test all components under src/

Figure 2: Cheat sheet for developing modular workflow components with Viash, including a sample Viash component (**left**) and common commands used throughout the various stages of a development cycle (**right**).

State of the field

The realm of bioinformatics workflow management is evolving rapidly, with numerous frameworks and portability solutions emerging to address the escalating complexity and scale of data processing (Wratten et al., 2021). Viash positions itself uniquely in this landscape as a meta-framework, focusing on the creation of portable workflow modules.

Workflow frameworks can be broadly categorised into three broad categories: graphical, programmatic, and specification-based types. Graphical workflow frameworks such as Galaxy (Goecks et al., 2010) and KNIME (Fillbrunn et al., 2017) are user-friendly for non-coders, while programmatic workflow frameworks such as Nextflow (Di Tommaso et al., 2017), Snakemake (Köster & Rahmann, 2012), and WDL (https://openwdl.org) offer a DSL or programming library for developers. Specification-based workflow frameworks such as CWL (Crusoe et al., 2022) lie somewhere in between. These allow one to describe and execute workflows with specification files (e.g., in YAML), and these specification files can be constructed using graphical or programmatic interfaces.

Portability solutions are critical for ensuring reproducibility. These can be divided into package managers like Conda for automated installation of versioned software, and containerization tools like Docker (https://www.docker.com) and Podman (Heon et al., 2018), which package and distribute software dependencies in a self-contained and platform-independent manner.

Viash's role in this landscape is to bridge these diverse tools, enabling more efficient and collaborative development in bioinformatics workflows.

Applications in bioinformatics

Ultimately, Viash aims to support pan-organisational and interdisciplinary research projects by simplifying collaborative development and maintenance of (complex) workflows. While Viash is generally applicable to any field where scalable and reproducible data processing workflows are needed, one field where it is particularly useful is in bioinformatics since it supports most of the commonly used technologies in this field, namely Bash, Python, R, Docker, and Nextflow.

The NeurIPS2021 competition organised by OpenProblems demonstrates the practical value of Viash (Luecken et al., 2021). As part of the preparation for the competition, a pilot benchmark was implemented to evaluate and compare the performance of a few baseline methods (Figure 3A). By pre-defining the input-output interfaces of several types of components (e.g., dataset loaders, baseline methods, control methods, metrics), developers from different organisations across the globe could easily contribute Viash components to the workflow (Figure 3B). Since Viash automatically generates Docker containers and Nextflow workflows from the metadata provided by component developers, developers could contribute components whilst making use of their programming environment of choice without needing to have any expert knowledge of Nextflow or Amazon EC2. Thanks to the modularity of Viash components, the same components used in running a pilot benchmark are also used by the evaluation worker of the competition website itself. As such, the pilot benchmark also serves as an integration test of the evaluation worker.

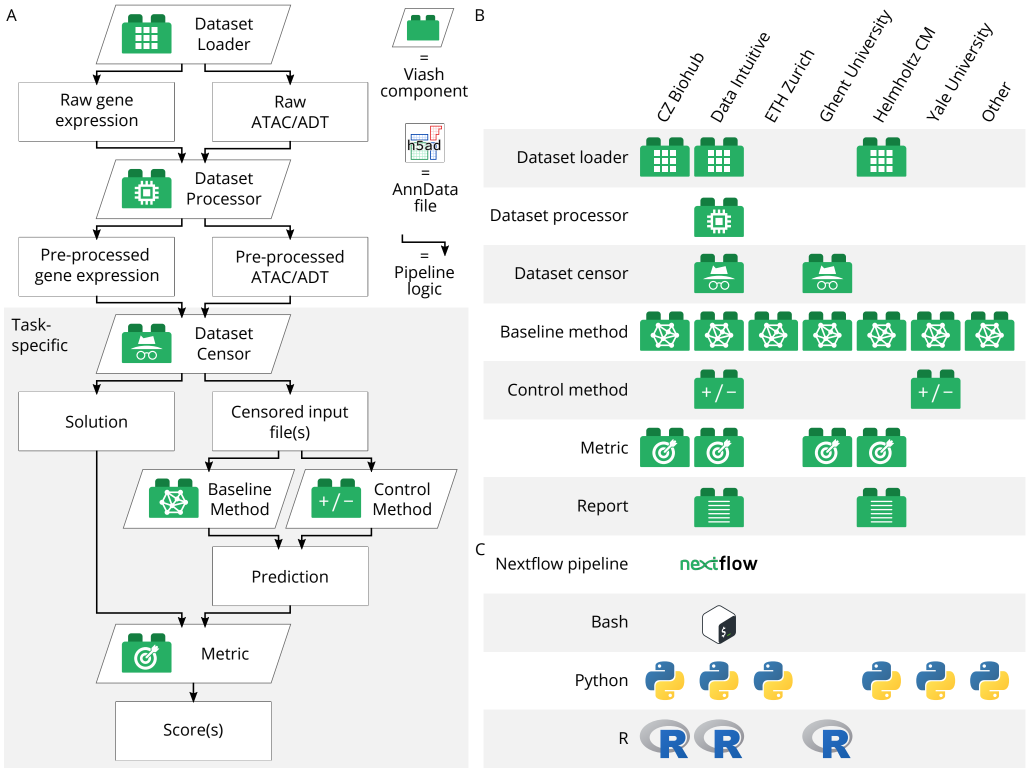


Figure 3: A recent NeurIPS competition for multimodal data integration (Luecken et al., 2021) demonstrates the practical value of Viash by using Bash, R, Python, Docker, Nextflow, Viash, and Amazon EC2 as core technologies to run a pilot benchmark. **A:** The pilot benchmark workflow consists of several types of components, each of which had strict predefined input-output interfaces. **B:** Comparing which organisations contributed one or more Viash components to the workflow demonstrates that Viash allows multiple organisations to participate in developing a workflow collaboratively. Developers are encouraged to implement components in their preferred scripting language. Thanks to the modularity provided by Viash, sewing together multiple components into a Nextflow workflow can be left up to a few developers, without requiring all collaborators to have expert knowledge regarding infrastructure-specific technologies. Note: this visualisation pertains to one aspect of organising the NeurIPS competition, and does not at all reflect the overall efforts made by any party.

Discussion

Viash is under active development. Active areas of development include expanded compatibility between Viash and other technologies (i.e., additional scripting languages, containerisation frameworks, and workflow frameworks), and ease-of-use functionality for developing and managing large catalogues of Viash components (e.g., simplified continuous integration, allowing project-wide settings, automating versioned releases).

We appreciate and encourage contributions to or extensions of Viash. All source code is available under a GPL-3 licence on Github at github.com/viash-io/viash. Extensive user documentation is available at viash.io. Requests for support or expanded functionality can be addressed to the corresponding authors.

References

Amezquita, R. A., Lun, A. T. L., Becht, E., Carey, V. J., Carpp, L. N., Geistlinger, L., Marini, F., Rue-Albrecht, K., Risso, D., Soneson, C., Waldron, L., Pagès, H., Smith, M. L., Huber, W., Morgan, M., Gottardo, R., & Hicks, S. C. (2019). Orchestrating single-cell analysis with bioconductor. *Nature Methods*, *17*(2), 137–145. https://doi.org/10.1038/s41592-019-0654-x

Crusoe, M. R., Abeln, S., Iosup, A., Amstutz, P., Chilton, J., Tijanić, N., Ménager, H., Soiland-Reyes, S., Gavrilović, B., Goble, C., & Community, T. C. (2022). Methods included: Standardizing computational reuse and portability with the common workflow language. *Communications of the ACM*, *65*(6), 54–63. https://doi.org/10.1145/3486897

Di Tommaso, P., Chatzou, M., Floden, E. W., Barja, P. P., Palumbo, E., & Notredame, C. (2017). Nextflow enables reproducible computational workflows. *Nature Biotechnology*, *35*(4), 316–319. https://doi.org/10.1038/nbt.3820

Ewels, P. A., Peltzer, A., Fillinger, S., Patel, H., Alneberg, J., Wilm, A., Garcia, M. U., Di Tommaso, P., & Nahnsen, S. (2020). The nf-core framework for community-curated bioinformatics pipelines. *Nature Biotechnology*, *38*(3), 276–278. https://doi.org/10.1038/s41587-020-0439-x

Fillbrunn, A., Dietz, C., Pfeuffer, J., Rahn, R., Landrum, G. A., & Berthold, M. R. (2017). KNIME for reproducible cross-domain analysis of life science data. *Journal of Biotechnology*, *261*, 149–156. https://doi.org/10.1016/j.jbiotec.2017.07.028

Goecks, J., Nekrutenko, A., Taylor, J., & Galaxy Team, T. (2010). Galaxy: A comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, *11*(8), R86. https://doi.org/10.1186/gb-2010-11-8-r86

Heon, M. et. al, Walsh, D., Baude, B., Mohnani, U., Cui, A., Sweeney, T., Scrivano, G., Evich, C., Rothberg, V., Trmač, M., Honce, J., Wang, Q., Mandvekar, L., Reber, A., Santiago, E., Grunert, S., Dahyabhai, N., Bjorklund, A., Kushwaha, K., … Podman Community. (2018). *Podman - : A tool for managing OCI containers and pods*. Zenodo. https://doi.org/10.5281/ZENODO.4735634

Heumos, L., Schaar, A. C., Lance, C., Litinetskaya, A., Drost, F., Zappia, L., Lücken, M. D., Strobl, D. C., Henao, J., Curion, F., Aliee, H., Ansari, M., Badia-i-Mompel, P., Büttner, M., Dann, E., Dimitrov, D., Dony, L., Frishberg, A., He, D., … Theis, F. J. (2023). Best practices for single-cell analysis across modalities. *Nature Reviews Genetics*, *24*(8), 550–572. https://doi.org/10.1038/s41576-023-00586-w

Köster, J., & Rahmann, S. (2012). Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, *28*(19), 2520–2522. https://doi.org/10.1093/bioinformatics/bts480

Luecken, M. D., Burkhardt, D. B., Cannoodt, R., Lance, C., Agrawal, A., Aliee, H., Chen, A. T., Deconinck, L., Detweiler, A. M., Granados, A. A., Huynh, S., Isacco, L., Kim, Y. J., Klein, D., KUMAR, B. D., Kuppasani, S., Lickert, H., McGeever, A., Mekonen, H., … Bloom, J. M. (2021). A sandbox for prediction and integration of DNA, RNA, and proteins in single cells. *Thirty-Fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. https://openreview.net/forum?id=gN35BGa1Rt

Luecken, M. D., & Theis, F. J. (2019). Current best practices in single-cell RNA-seq analysis: A tutorial. *Molecular Systems Biology*, *15*(6). https://doi.org/10.15252/msb.20188746

Wratten, L., Wilm, A., & Göke, J. (2021). Reproducible, scalable, and shareable analysis pipelines with bioinformatics workflow managers. *Nature Methods*, *18*(10), 1161–1168. https://doi.org/10.1038/s41592-021-01254-9