pyrpipe: a python package for RNA-Seq workflows

Urminder Singh[1,2,4*], Jing Li[1,2,4], Arun Seetharam[3], and Eve Syrkin Wurtele[1,2,4*]

[1]Bioinformatics and Computational Biology Program, Iowa State University, Ames, IA 50011, USA
[2]Center for Metabolic Biology, Iowa State University, Ames, IA 50011, USA
[3]Genome Informatics Facility, Iowa State University, Ames, IA 50011, USA
[4]Department of Genetics Development and Cell Biology, Iowa State University, Ames, IA 50011, USA

Implementing RNA-Seq analysis pipelines is challenging as data gets bigger and more complex. With the availability of terabytes of RNA-Seq data and continuous development of analysis tools, there is a pressing requirement for frameworks that allow for fast and efficient development, modification, sharing and reuse of workflows. Scripting is often used, but it has many challenges and drawbacks. We have developed a python package, python RNA-Seq Pipeliner (`pyrpipe`) that enables straightforward development of flexible, reproducible and easy-to-debug computational pipelines purely in python, in an object-oriented manner. `pyrpipe` provides high level APIs to popular RNA-Seq tools. Pipelines can be customized by integrating new python code, third-party programs, or python libraries. Researchers can create checkpoints in the pipeline or integrate `pyrpipe` into a workflow management system, thus allowing execution on multiple computing environments. `pyrpipe` produces detailed analysis, and benchmark reports which can be shared or included in publications. `pyrpipe` is implemented in python and is compatible with python versions 3.6 and higher. All source code is available at https://github.com/urmi-21/pyrpipe; the package can be installed from the source or from PyPi (https://pypi.org/project/pyrpipe). Documentation is available on Read the Docs (http://pyrpipe.rtfd.io).

Introduction

Since its inception, RNA-Seq has become the most widely used method to quantify transcript levels (1); terabytes of publicly available RNA-Seq data, encompassing multiple species, organs, genotypes, and conditions, is deposited in public repositories (2). Integrated analysis of aggregations of thousands of diverse RNA-Seq samples enables exploration of changes in gene expression over time and across different conditions (3).

A major challenge of processing thousands of RNA-Seq datasets, is implementing data processing pipelines in an efficient, modular, and reproducible manner. Most bioinformatics tools are standalone linux programs, executed via the shell; bioinformatic pipelines are usually written as shell, perl, or python scripts, which may be integrated with Makefiles (4, 5). Scripting, although powerful and flexible, can be difficult to develop, understand, maintain, and debug.

*To whom correspondence should be addressed. Email: usingh@iastate.edu, mash@iastate.edu

Here we present `pyrpipe`, a lightweight python package for bioinformatics researchers to code and execute RNA-Seq workflows in an object oriented manner. `pyrpipe` provides high-level APIs for 15 popular RNA-Seq analysis tools including a dedicated module to easily access and manage RNA-Seq data available from the NCBI-SRA database (2). Researchers can integrate into `pyrpipe` their own python code, third-party programs, and existing python libraries in a flexible, straight-forward way. No new *workflow* syntax specific to `pyrpipe` are required. `pyrpipe` meticulously logs information related to pipeline execution, providing extensive debugging resources. After each analysis, researchers can generate reports and summaries with all information necessary to reproduce the analysis.

`pyrpipe` is not a workflow management system, such as the popular Snakemake (6) or NextFlow (7), in that it does not scale jobs on clusters, or manage memory and parallel processing. However, `pyrpipe` is designed to be readily integrated into workflow management systems, providing a customizable framework for reproducible and scaleable pipelines.

Implementation

We developed `pyrpipe` to create an easy-to-use python framework for researchers to code, share, and reuse RNA-Seq analysis workflows. `pyrpipe` achieves this by providing: 1. high level APIs to popular RNA-Seq tools; 2. a general API to execute within python any shell command, enabling use of any bioinformatics tool; and 3. extensive logging details of the commands. We selected the Python platform because it is widely used, free, flexible, object-oriented, and has high-level data structures (8), (9), with a repository of $> 200,000$ packages and tools.

A. Object-oriented and modular design. We have taken an object oriented approach to implement `pyrpipe`, such that any RNA-Seq processing workflow can be intuitively executed by the researcher. `pyrpipe`'s modular design permits writing code that is easy to read, manage, and share. From a developer's perspective, modularity facilitates reuse and extensibility; new tools can be easily integrated into `pyrpipe`.

`pyprpipe` consists of highly cohesive modules (*sra*, *mapping*, *alignment*, *quant*, *qc*, *tools*) designed to capture steps integral to RNA-Seq analysis (Supplementary Table

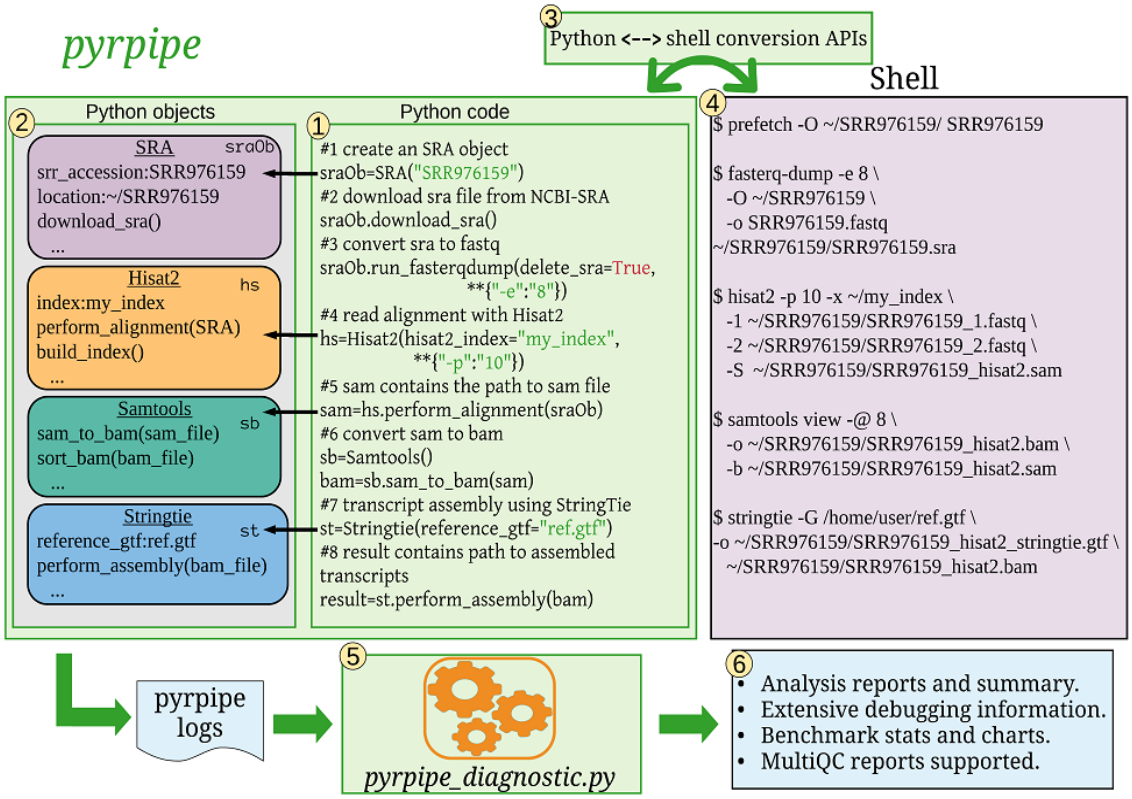


Fig. 1. A simple example demonstrating `pyrpipe`. The researcher imports `pyrpipe` and writes the code in python (Box 1). At each step in the python code, an *object* is created (Box 2). Each object encapsulates specific *methods* and *data*. The *pyrpipe_engine* module forwards commands to shell via the *subprocess* library, monitors status, and updates the logs continuously (Box 3). Shell commands are automatically constructed by the `pyrpipe` APIs but also can be explicitly provided by the user (Box 4). After execution, the *pyrpipe_diagnostic.py* module (Box 5) generates analyses and reports (Box 6) from the logs. `pyrpipe` is represented by green boxes.

1). The modules *pyrpipe_engine*, *pyrpipe_module* and *pyrpipe_diagnostic* contain multiple helper functions and classes.

Using classes to encapsulate various "tools" and "data" is the key concept in `pyrpipe`. For each module, we identified and implemented *abstract classes* to represent operations: access to NCBI-SRA, quality control, read alignment, transcript assembly and transcript quantification. To date, we have implemented 17 children classes providing APIs to specific RNA-Seq tools. (Supplementary Table 1 and Supplementary Fig. 1). Thus, "tools" can be easily accessed as objects, while ensuring that associated data and parameters are consistently accessible within that object.

A workflow is defined using *instances* or *objects* of these classes (Fig. 1). Once objects are created, they can be reused throughout the program, promoting faster development and code re-usability, for example, a Hisat2 object can be used to align reads from multiple SRA objects.

B. Flexibility in pipeline execution, debugging, reproducible analysis, and pipeline sharing. `pyrpipe` flexibility extends to choice of how the pipeline is designed to execute, and handle exceptions and errors. Researchers can create checkpoints in the pipeline, save the current `pyrpipe` *session*, and resume later. This is particularly useful for running different blocks of a workflow in different environments. For example, on a typical high performance computing (HPC) cluster, a researcher might use a dedicated data-transfer node to retrieve data from SRA and then use compute nodes for data processing.

`pyrpipe` has automatic logging features for efficient error detection and reports (Fig. 1). *pyrpipe_diagnostic.py* module includes a logger object and logs all executed commands and their outputs. Environment information, such as operating system and Python version, along with version and path information for each program used within the pipeline, are also logged. `pyrpipe` logs are saved in JavaScript Object Notation (JSON) format for easy parsing by other programs (Supplementary Table 2).

The *pyrpipe_diagnostic.py* module generates comprehensive reports about the analysis, benchmark comparisons, shell commands, reports for debugging, and MultiQC reports (10). These reports, along with the python scripts, can be shared or included with publications for reproducible research.

Example usage

Transcript assembly using pyrpipe. . We demonstrate `pyrpipe`'s usability by processing *Zea mays* RNA-Seq data available through NCBI-SRA (2). The workflow is explained in following steps:

1. Importing `pyrpipe`: To use `pyrpipe`, we need to import it in current python session. Lines 1-5 imports the `pyrpipe` modules in python. Line 7 initializes a list of SRR accessions used in this examples. Line 10

initializes *workingDir* variable which contains the path where all data will be downloaded.

```
1  from pyrpipe import sra
2  from pyrpipe import mapping
3  from pyrpipe import assembly
4  from pyrpipe import qc
5  from pyrpipe import tools
6
7  runs=['SRR3098746',
8      'SRR3098745',
9      'SRR3098744']
10 workingDir='maize_out'
```

2. Downloading raw data: First we created *SRA* objects corresponding to each SRR accession (Line 13). To download raw data from NCBI-SRA, we used the *download_fastq()* (line 14) to download reads in fastq format. We used Trim Galore object (line 19) and performed quality filtering (line 23). Parameters were passed to Trim Galore object as a dict (line 19).

```
11 sraObs=[]
12 for x in runs:
13     ob=sra.SRA(x,workingDir)
14     if ob.download_fastq():
15         sraObs.append(ob)
16
17 #create a Trimgalore object
18 tgOptions={"--cores": "10"}
19 tg=qc.Trimgalore()
20
21 for ob in sraObs:
22     #perform qc using trim galore
23     ob.perform_qc(tg,**tgOpts)
```

3. Mapping reads and transcript assembly: We create an object to use STAR aligner (line 32) and StringTie (line 34). Then we process all SRA objects in a loop (lines 37-39). First, mapping the reads to the genome using STAR (line 38) and the performing transcript assembly with StringTie (line 39).

```
24 starParams={'--outFilterType':
25             'BySJout',
26             '--outSAMtype':
27             'BAM SortedByCoordinate'
28             }
29
30 star=mapping.Star(index='index')
31 st=assembly.Stringtie()
32
33 for ob in sraObs:
34     bam=star.perform_alignment(ob,
35     **starParams)
36     st.perform_assembly(reference_gtf=
37         'ref.gtf',bam)
```

Case studies

C. Prediction of long non-coding RNAs (lncRNAs) in RNA-Seq *Zea mays* by supplementing `pyrpipe` with a third-party tool.. We downloaded RNA-Seq data from SRA, quality filtered using Trim Galore (11), aligned reads to the *Maize* genome using STAR (12) and transcripts were assembled using StringTie (13). Then, we used a third party tool, (PLncPRO (14)), to predict lncRNAs, and as-yet-unannotated mRNAs, in the assembled transcripts. Case study: https://github.com/urmi-21/pyrpipe/tree/master/case_studies/Maize_lncRNA_prediction.

D. *Arabidopsis thaliana* transcript assembly using `pyrpipe` checkpoints. We downloaded raw read RNA-Seq data for Arabidopsis from SRA, performed quality control using BBDuk (15), aligned reads to the genome using Hisat2 (16) and assembled transcripts using StringTie (13). Case study: https://github.com/urmi-21/pyrpipe/tree/master/case_studies/Athaliana_transcript_assembly

E. Integrating `pyrpipe` scripts within a workflow management system. We embedded *pyrpipe* into the Snakemake workflow management system (6), and used it to download human RNA-Seq data with SRAtools, quality filter the data with BBDuk (15), align reads with Hisat2 (12), assemble transcripts with StringTie (13) and Cufflinks (17), and merge the multiple assemblies with Mikado (18). Case study: https://github.com/urmi-21/pyrpipe/tree/master/case_studies/Human_annotation_snakemake

F. Prediction of *Zea mays* orphan genes. In this case study we used ten diverse *Zea mays* RNA-Seq samples from NCBI-SRA to identify transcripts that would encode candidate species-specific("orphan") genes. Supplementary Fig. 2 shows the workflow. `pyrpipe` scripts, downstream analysis code, and data is available at https://github.com/lijing28101/maize_pyrpipe. The results are discussed below.

Results

The orphan genes of the current high-quality genome of *Zea mays* B73 had not been systematically annotated, thus, we examined the trends among orphan vs non-orphan transcripts of ten RNA-Seq runs from this line. Our analysis pipeline for this RNA-Seq data identified a total of 60,999 distinct transcripts that contained an ORF greater than 150 nt. A subset of these will represent protein-coding genes; others will be lncRNAs or expression products that do not represent genes. 6,306 of these transcripts contained ORFs whose translated product shows no homology to proteins of any other species ("orphan-coding transcripts"). Fig. 2 shows the transcript length, and GC content distribution for these orphan-coding and non-orphan-coding transcripts. The length of orphan-coding transcripts is shorter than for non-orphan-coding tran-

scripts. However, GC content distribution is indistinguishable between orphans and non-orphans. These trends are quite similar to those of annotated orphan and non-orphan genes from *Arabidopsis thaliana* (19), although the median number of exons reported in orphan-coding transcripts in *A. thaliana* is one, versus that in *Z. mays* of two.

We compared the expression level of orphan and non- orphan transcripts within each RNA-Seq sample (Fig. 3). In each of the 10 runs analyzed, median expression of orphan-coding transcripts is much lower as compared to median expression of non-orphan-coding transcripts. However, in each run but one, some orphan-coding transcripts are highly expressed.

Conclusion

The `pyrpipe` package allows researchers to code and implement RNA-Seq workflows in an object oriented manner, purely using python. `pyrpipe` can be integrated into workflow management systems or used directly. Access to NCBI-SRA is automated, such that researchers can readily retrieve raw RNA-Seq data. The downloaded data and data files are automatically managed, and consistently accessed through *SRA* objects. Researcher need not keep track of data files or their paths, as these are integrated with `pyrpipe` objects. `pyrpipe` workflows can be modified using python's control flow abilities and a user can create complex, reproducible, workflow structures. Any third party tool or script can be integrated into `pyrpipe` for additional data processing capability. `pyrpipe` logs and reports enable debugging and reproducibility. textttpyrpipe workflows provide a clear record for publications.

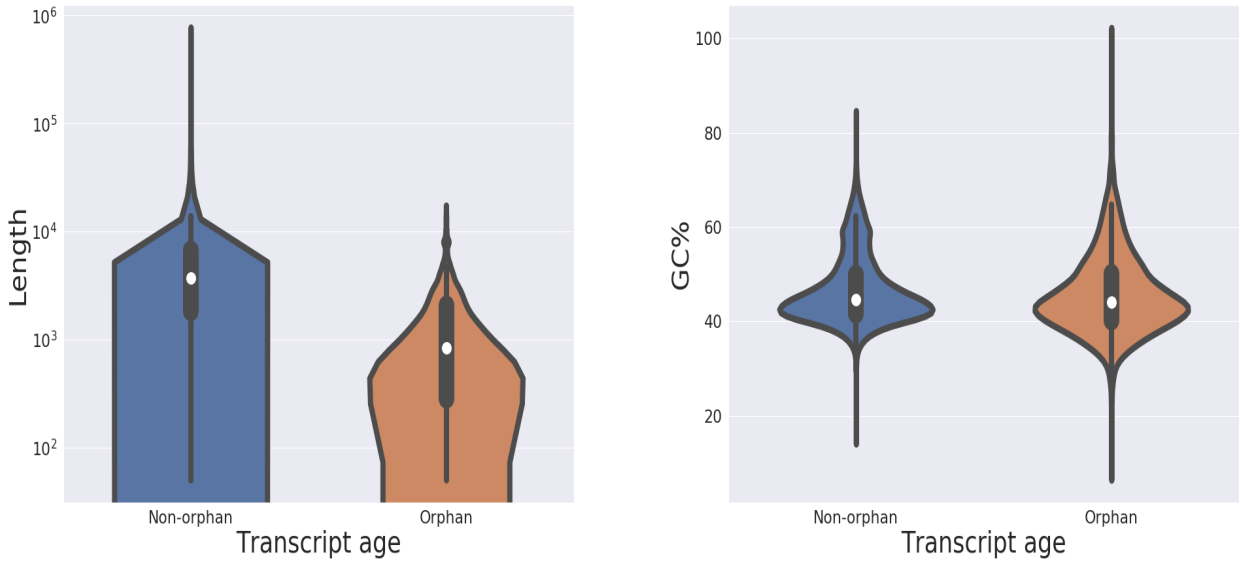
`pyrpipe` will appeal to researchers who are looking for simple, fast way to deploy large RNA-Seq processing pipelines. Straightforward implementation, execution and sharing of RNA-Seq workflows makes it an ideal choice for researchers with less computational expertise.

Funding

This work is funded in part by National Science Foundation grant IOS 1546858, Orphan Genes: An Untapped Genetic Reservoir of Novel Traits, and by the Center for Metabolic Biology, Iowa State University.

Bibliography

1. Peipei Li, Yongjun Piao, Ho Sun Shon, and Keun Ho Ryu. Comparing the normalization methods for the differential analysis of illumina high-throughput rna-seq data. *BMC bioinformatics*, 16(1):347, 2015.
2. Yuichi Kodama, Martin Shumway, and Rasko Leinonen. The sequence read archive: explosive growth of sequencing data. *Nucleic acids research*, 40(D1):D54–D56, 2011.
3. Urminder Singh, Manhoi Hur, Karin Dorman, and Eve Syrkin Wurtele. MetaOmGraph: a workbench for interactive exploratory data analysis of large expression datasets. *Nucleic Acids Research*, 01 2020. ISSN 0305-1048. doi: 10.1093/nar/gkz1209. gkz1209.
4. Jeremy Leipzig. A review of bioinformatic pipeline frameworks. *Briefings in bioinformatics*, 18(3):530–536, 2017.
5. Ola Spjuth, Erik Bongcam-Rudloff, Guillermo Carrasco Hernández, Lukas Forer, Mario Giovacchini, Roman Valls Guimera, Aleksi Kallio, Eija Korpelainen, Maciej M Kańduła, Milko Krachunov, et al. Experiences with workflows for automating data-intensive bioinformatics. *Biology direct*, 10(1):43, 2015.
6. Johannes Köster and Sven Rahmann. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 2012.
7. Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. Nextflow enables reproducible computational workflows. *Nature biotechnology*, 35(4):316, 2017.
8. Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. Tensorly: Tensor learning in python. *The Journal of Machine Learning Research*, 20(1):925–930, 2019.
9. Hans Petter Langtangen, Timothy J Barth, and Michael Griebel. *Python scripting for computational science*, volume 3. Springer, 2006.
10. Philip Ewels, Måns Magnusson, Sverker Lundin, and Max Käller. Multiqc: summarize analysis results for multiple tools and samples in a single report. *Bioinformatics*, 32(19):3047–3048, 2016.
11. Felix Krueger. Trim galore. *A wrapper tool around Cutadapt and FastQC to consistently apply quality and adapter trimming to FastQ files*, 2015.
12. Alexander Dobin, Carrie A Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R Gingeras. Star: ultrafast universal rna-seq aligner. *Bioinformatics*, 29(1):15–21, 2013.
13. Mihaela Pertea, Geo M Pertea, Corina M Antonescu, Tsung-Cheng Chang, Joshua T Mendell, and Steven L Salzberg. Stringtie enables improved reconstruction of a transcriptome from rna-seq reads. *Nature biotechnology*, 33(3):290, 2015.
14. Urminder Singh, Niraj Khemka, Mohan Singh Rajkumar, Rohini Garg, and Mukesh Jain. PLncPRO for prediction of long non-coding rnas (lncrnas) in plants and its application for discovery of abiotic stress-responsive lncrnas in rice and chickpea. *Nucleic acids research*, 45(22):e183–e183, 2017.
15. B Bushnell. Bbtools software package. *URL http://sourceforge. net/projects/bbmap*, 2014.
16. 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, 2019.
17. Cole Trapnell, Brian A Williams, Geo Pertea, Ali Mortazavi, Gordon Kwan, Marijke J Van Baren, Steven L Salzberg, Barbara J Wold, and Lior Pachter. Transcript assembly and quantification by rna-seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature biotechnology*, 28(5):511, 2010.
18. Luca Venturini, Shabhonam Caim, Gemy George Kaithakottil, Daniel Lee Mapleson, and David Swarbreck. Leveraging multiple transcriptome assembly methods for improved gene structure annotation. *GigaScience*, 7(8):giy093, 2018.
19. Zebulun W Arendsee, Ling Li, and Eve Syrkin Wurtele. Coming of age: orphan genes in plants. *Trends in plant science*, 19(11):698–708, 2014.
20. Stephen Sherry and Chunlin Xiao. Ncbi sra toolkit technology for next generation sequence data. In *Plant and Animal Genome XX Conference (January 14-18, 2012). Plant and Animal Genome*, 2012.
21. Ben Langmead and Steven L Salzberg. Fast gapped-read alignment with bowtie 2. *Nature methods*, 9(4):357, 2012.
22. Manfred G Grabherr, Brian J Haas, Moran Yassour, Joshua Z Levin, Dawn A Thompson, Ido Amit, Xian Adiconis, Lin Fan, Raktima Raychowdhury, Qiandong Zeng, et al. Trinity: reconstructing a full-length transcriptome without a genome from rna-seq data. *Nature biotechnology*, 29(7):644, 2011.
23. Nicolas L Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic rna-seq quantification. *Nature biotechnology*, 34(5):525, 2016.
24. Rob Patro, Geet Duggal, Michael I Love, Rafael A Irizarry, and Carl Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature methods*, 14(4): 417, 2017.
25. Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, and Richard Durbin. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079, 2009.
26. Daniel Mapleson, Luca Venturini, Gemy Kaithakottil, and David Swarbreck. Efficient and accurate detection of splice junctions from rna-seq with portcullis. *GigaScience*, 7(12): giy131, 2018.
27. Zhengtao Xiao, Rongyao Huang, Xudong Xing, Yuling Chen, Haiteng Deng, and Xuerui Yang. De novo annotation and characterization of the translatome with ribosome profiling data. *Nucleic acids research*, 46(10):e61–e61, 2018.
28. Benjamin Buchfink, Chao Xie, and Daniel H Huson. Fast and sensitive protein alignment using diamond. *Nature methods*, 12(1):59, 2015.



(a) Length distribution

(b) %GC content

Fig. 2. Comparison of length, and %GC content among transcripts whose ORFs are orphan-coding and non-orphan proteins, as identified by our analysis pipeline.

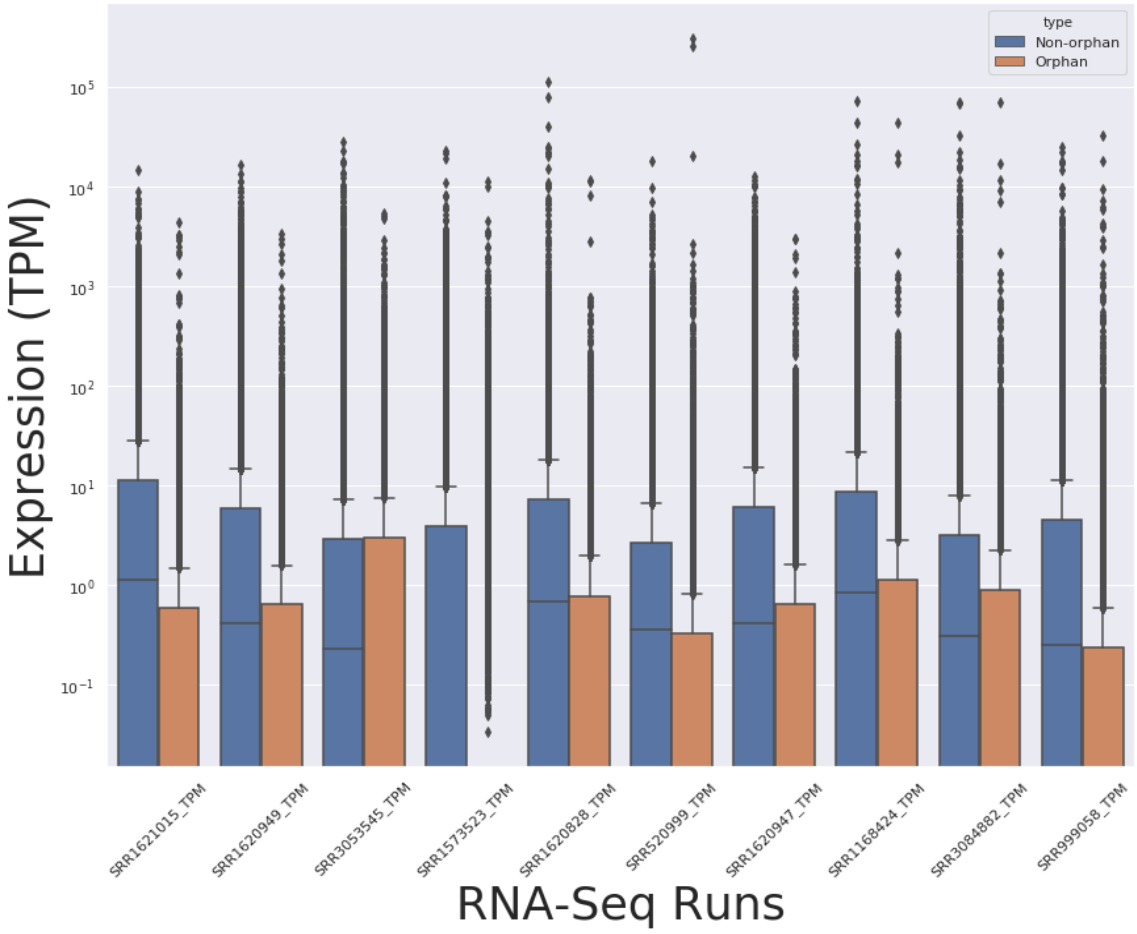


Fig. 3. Box plots showing expression level (TPM) of expressed orphan and non-orphan protein coding transcripts in 10 RNA-Seq runs.

Supplementary Information.

Module name	Class name	API for	Purpose
sra	SRA	sra-tools (20)	Access NCBI-SRA database
mapping	Hisat2	Hisat2 (16)	Read alignment
	Star	STAR (12)	Read alignment
	Bowtie2	Bowtie2 (21)	Read alignment
assembly	Stringtie	StringTie (13)	Transcript assembly
	Cufflinks	Cufflinks (17)	Transcript assembly
	Trinity	Trinity (22)	Transcript assembly
quant	Kallisto	Kallisto (23)	Transcript quantification
	Salmon	Salmon (24)	Transcript quantification
qc	Trimgalore	Trim Galore (11)	Quality control
	BBDuk	BBDuk (15)	Quality control
tools	Samtools	SAMtools (25)	Processing read alignments
	Portcullis	Portcullis (26)	Detect splice junctions
	Mikado	Mikado (18)	Integrate multiple RNA-seq assemblies
	Ribocode	RiboCode (27)	Analyze ribosome profiling data
	Diamond	Diamond (28)	Fast homology search
	TransDecoder	TransDecoder	Identify candidate coding regions

Table 1. Currently implemented *pyrpipe* modules. Each module contain multiple classes containing APIs for different RNA-Seq tools.

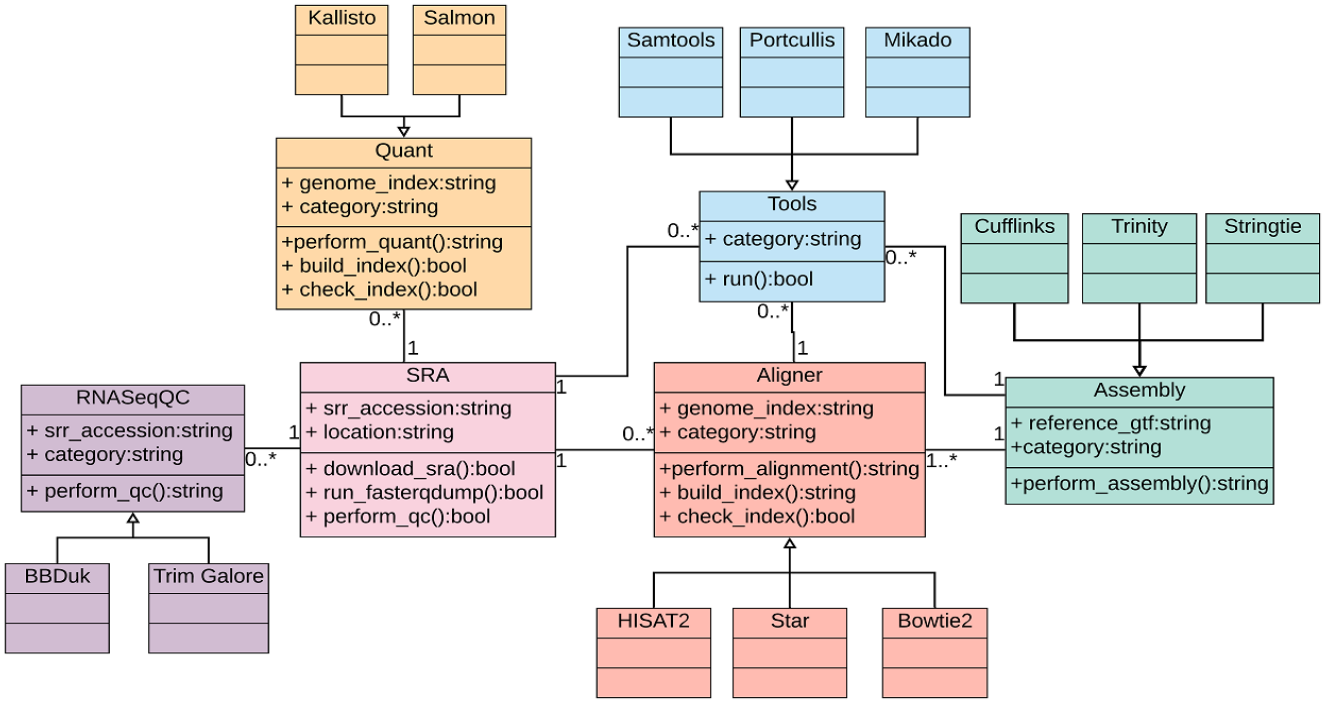


Fig. 1. A UML class diagram showing `pyrpipe's` classes and relationships among them. Classes in the same `pyrpipe` module share the same color.

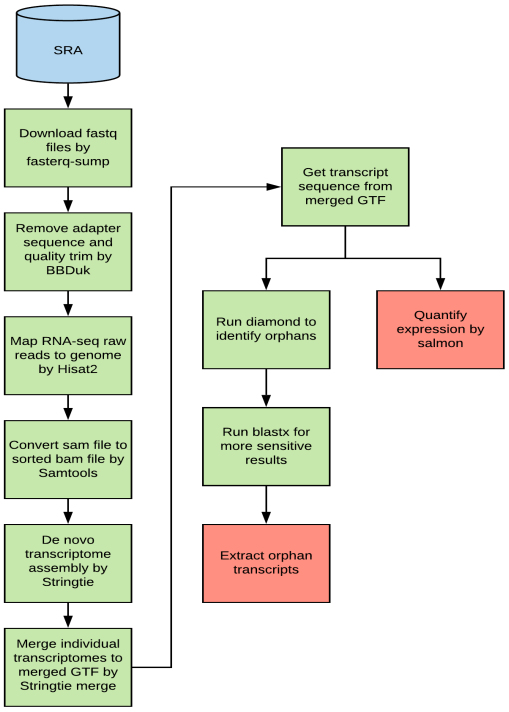


Fig. 2. A flowchart showing the pipeline implemented in `pyrpipe` to identify potentially orphan coding transcripts in *Zea mays*.

Key	**Description**
cmd	shell command executed
starttime	Time at the start of execution
runtime	Total runtime
exitcode	The return code
stdout	stdout returned by the program
stderr	stderr returned by the program
objectid	Id of an object used with the command
commandname	Name of the command
python	Python version
os	Operating system
cpu	CPU information
syspath	Python's sys.path
sysmodules	Python's sys.modules
name	Name of the program executed
version	Version of the program
path	Path to the program on disk

Table 2. Table showing description of the JSON *keys* stored in `pyrpipe` logs