Open Problems in Single-Cell Analysis

Creating a benchmark MVP in 60 minutes 😬

The core team

Daniel B. Burkhardt, PhD.

Co-founder

Machine Learning Scientist at Cellarity

Robrecht Cannoodt, PhD

Core Developer

Data Science Engineer at Data Intuitive

Scott Gigante, PhD

Co-founder

Founding Engineer at Sightline Climate

Christopher Lance

PhD Student at Helmholtz Zentrum München

Malte D. Luecken, D.Phil

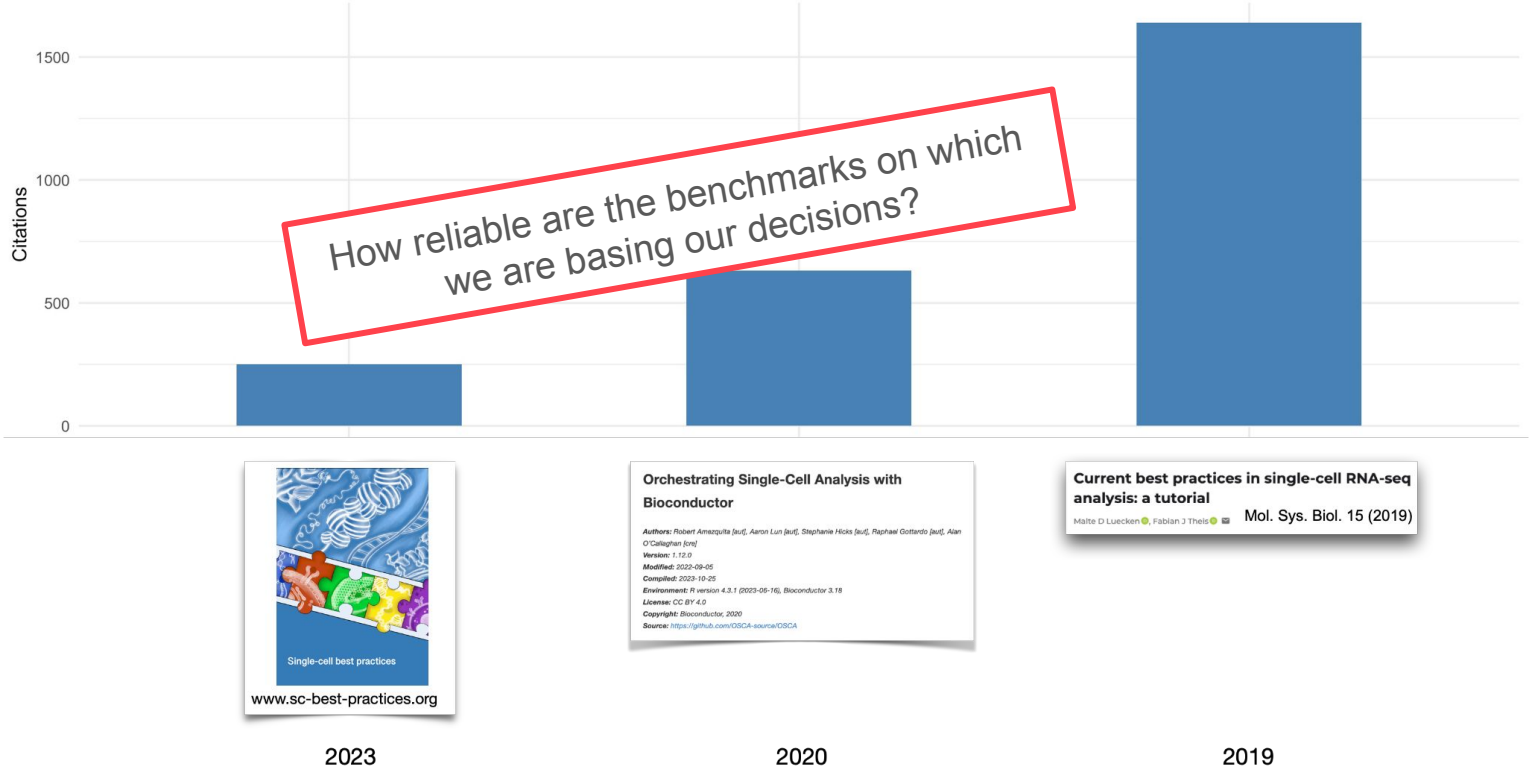
Co-founder

Group Leader at Helmholtz Zentrum München

Angela O. Pisco, PhD

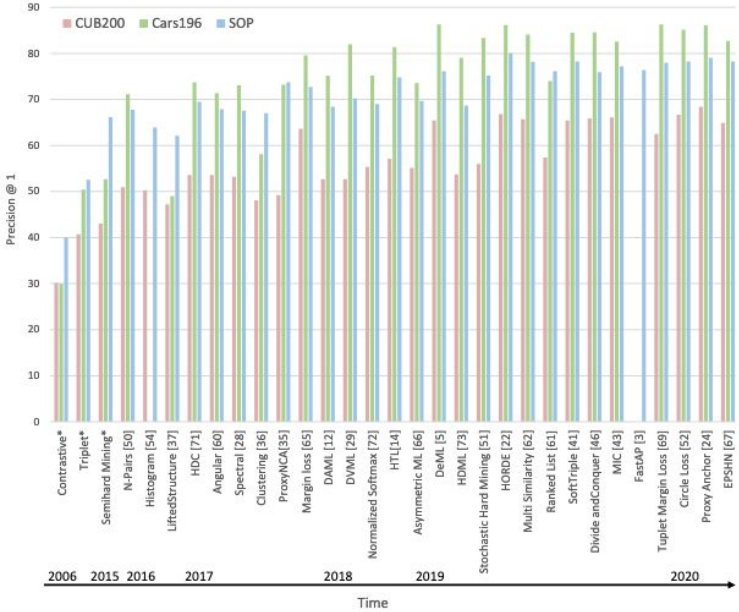
Director of Computational Biology at insitro

Benchmarking is driving analysis decisions in single-cell genomics

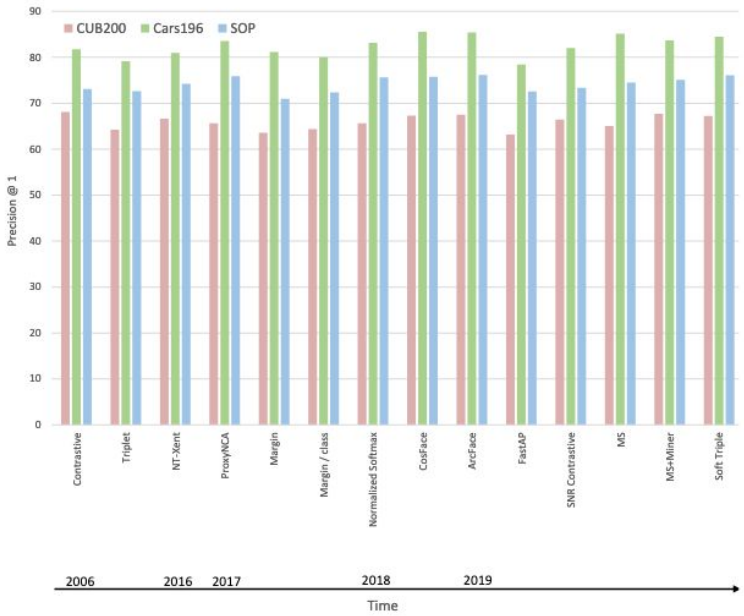


Luecken and Theis, Mol Sys Biol (2019); Heumos, Schaar, et al., Nat Biotech (2023);

Decentralized benchmarking exaggerates improvements



(a) The trend according to papers

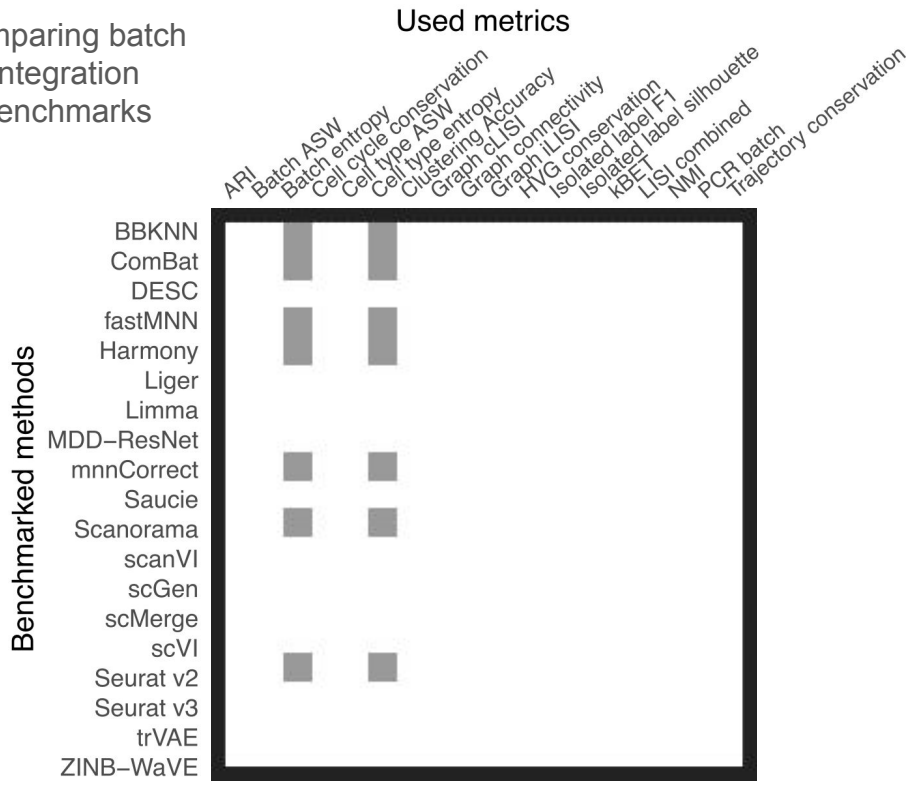


(b) The trend according to reality

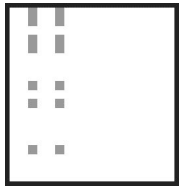
Musgrave et al,. A Metric learning reality check, arXiv (2020)

Single-cell benchmarks are hard to compare

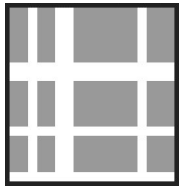
Comparing batch integration benchmarks



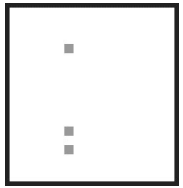
Chazarra-Gil et al., NAR (2021)



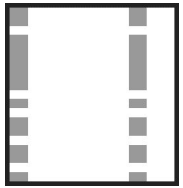
Chazarra-Gil et al., NAR (2021)



Luecken et al., Nat Meth (2022)

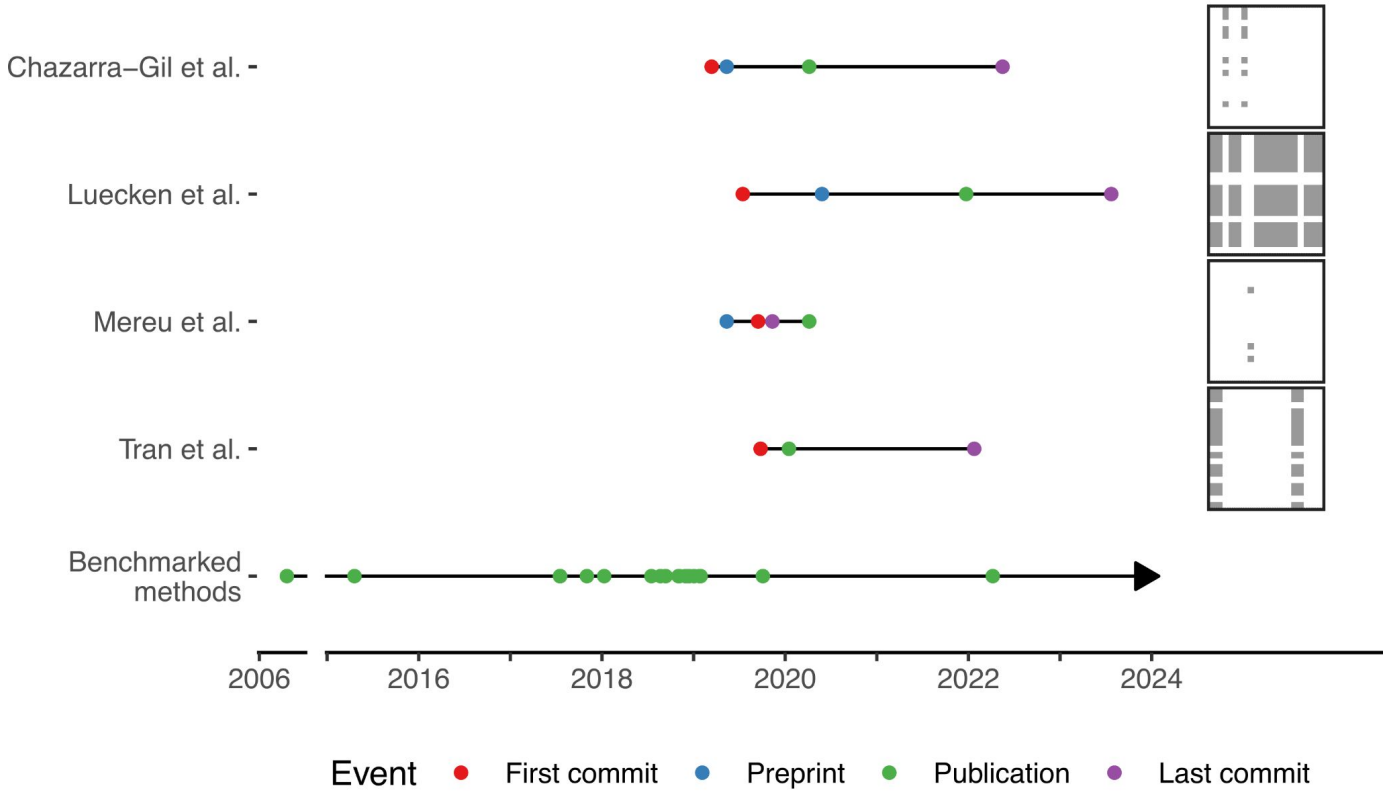


Mereu et al., Nat Biotech (2020)

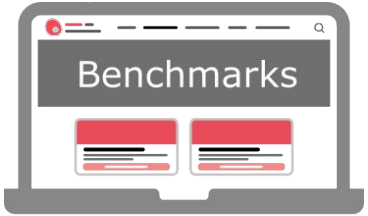


Tran et al., Genome Biol (2021)

Benchmarks inevitably age

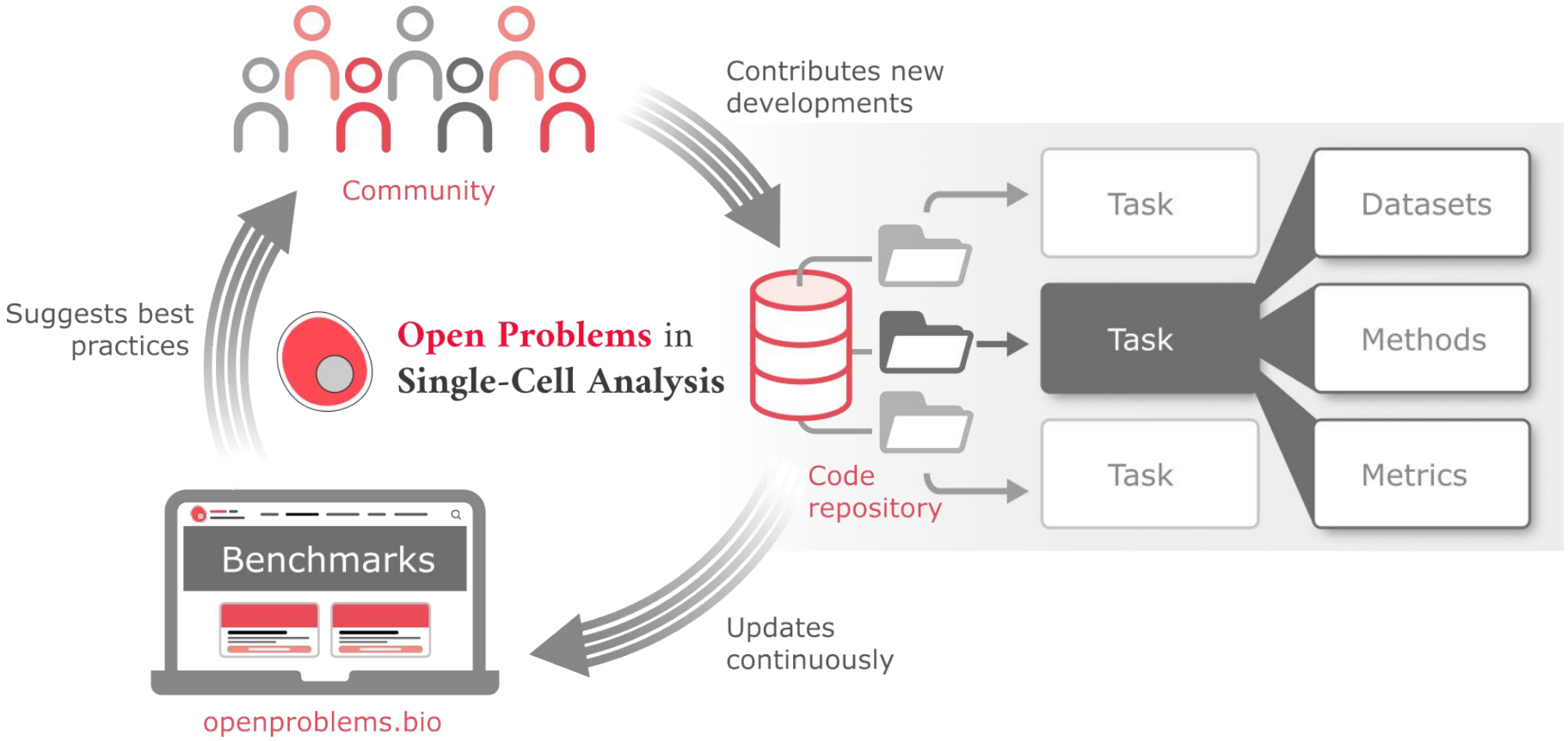


Addressing benchmarking limitations in Open Problems



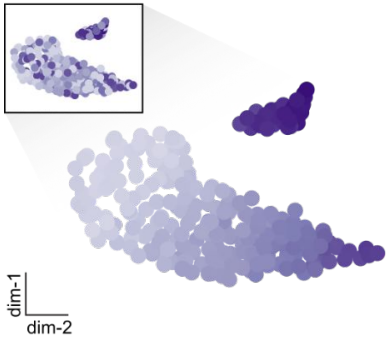
	Code repository	Community	openproblems.bio
Centralization	✓		
Consistency	✓		
Longevity	✓	✓	
Accessibility			✓

The Open Problems setup

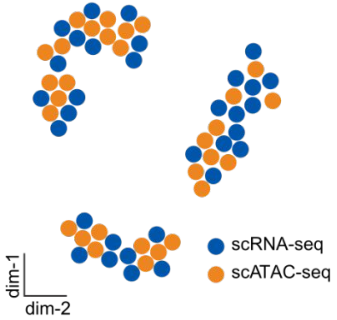


Community input defines standardized benchmarks

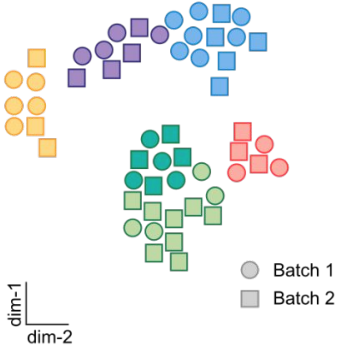
Data denoising



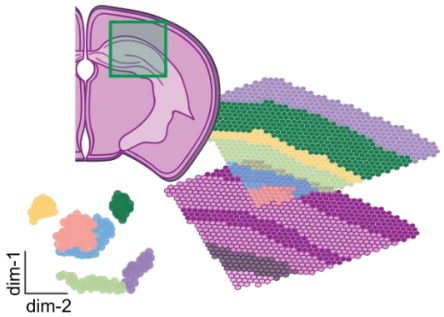
Multimodal integration



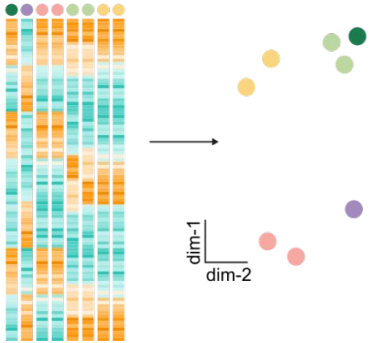
Batch integration



Spatial deconvolution

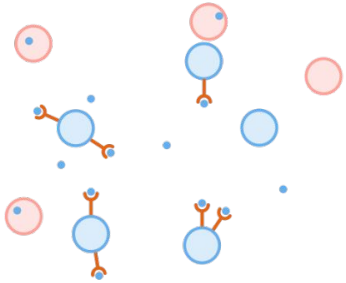


Dimensionality reduction

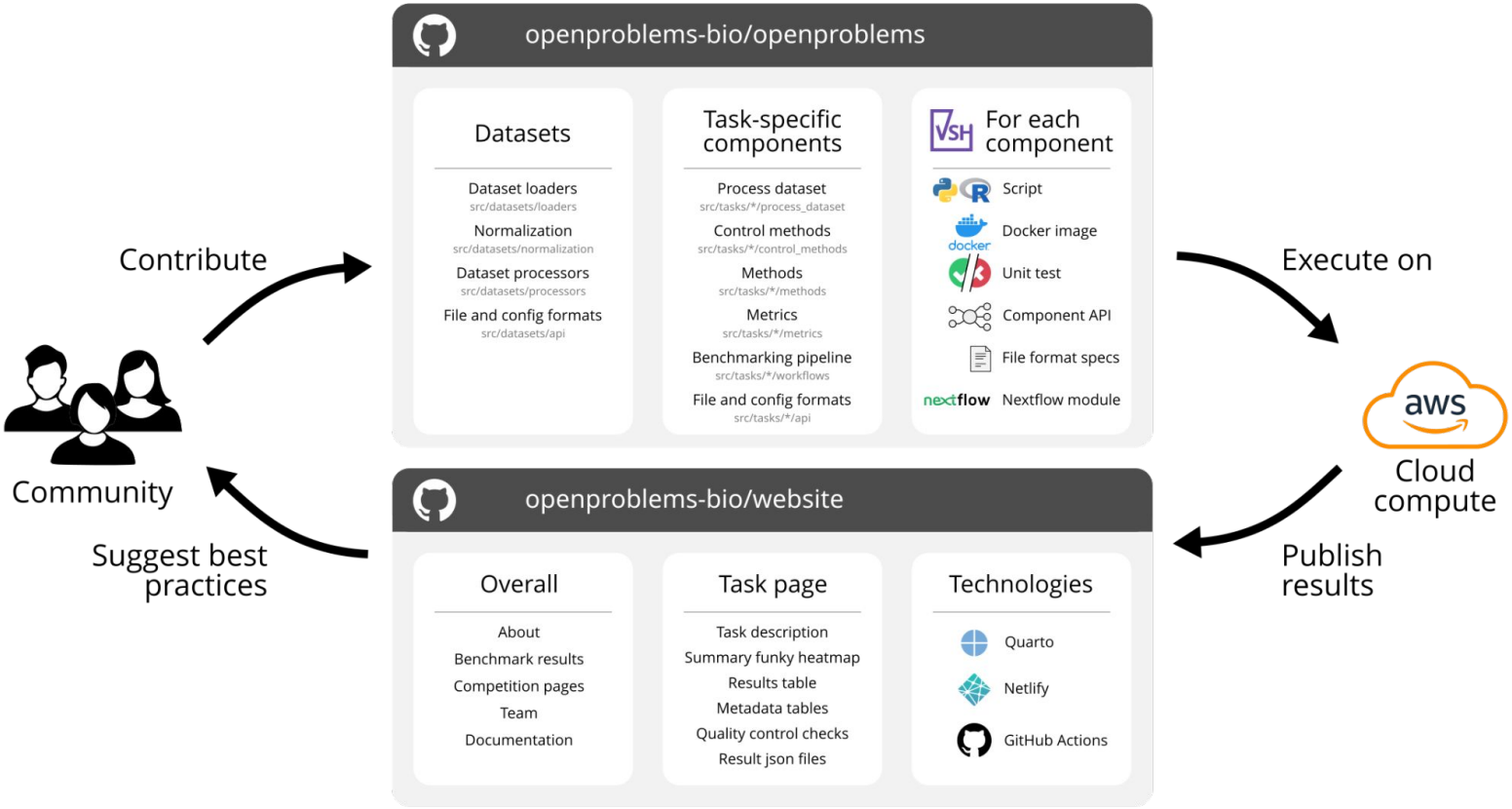


Label projection

Cell-cell communication



A look under the hood…

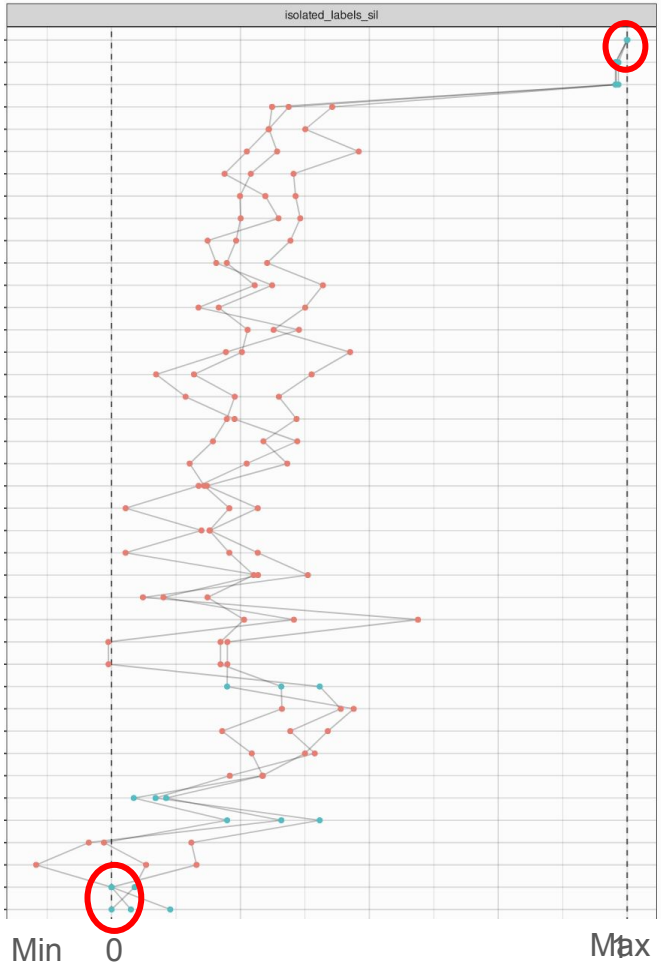


Accessibility: Ensuring comparable results across metrics via normalization

Challenge: metrics have different scales

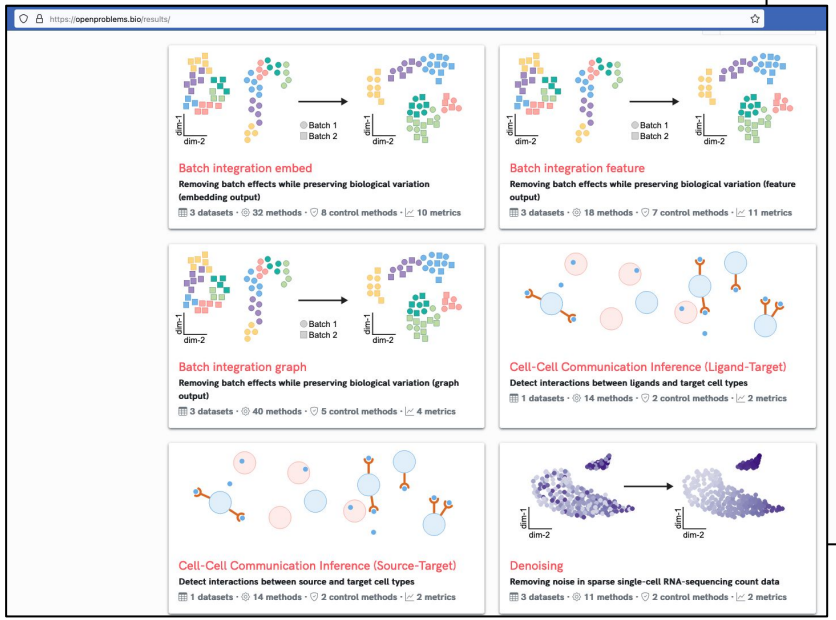
Solution: Normalization

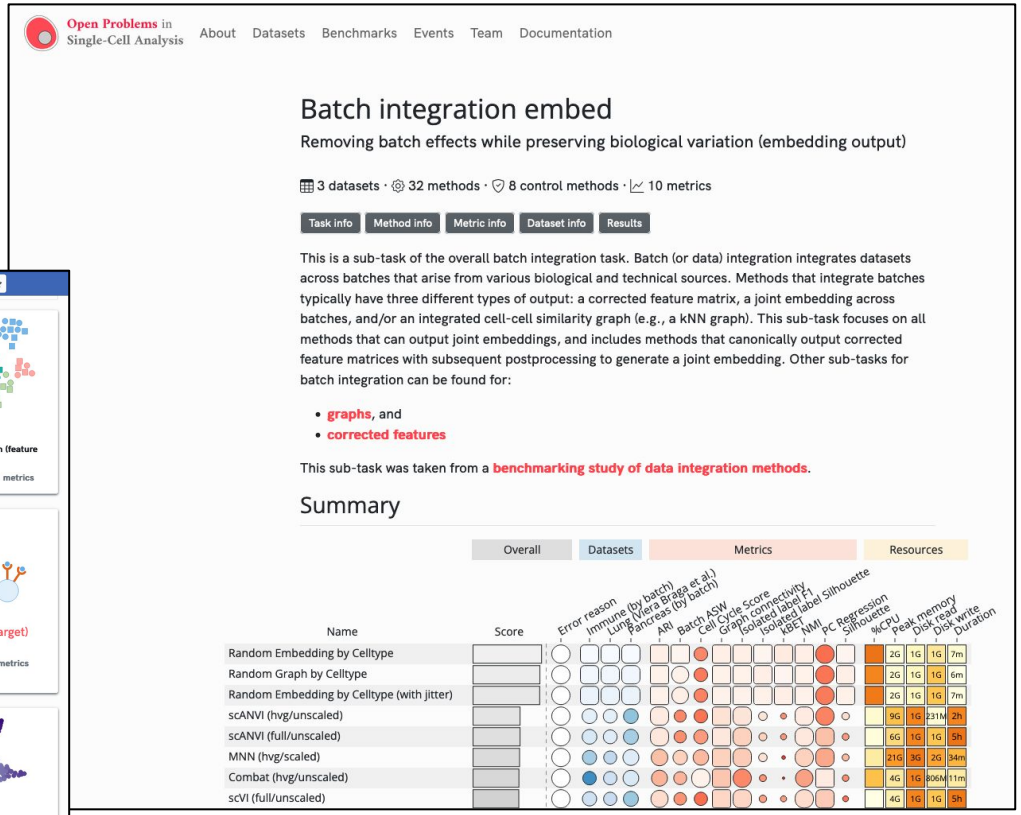
- Add (several) negative and positive baseline methods
  - E.g., random integration & cell type encoding as embedding
- Scale metric outputs to 0 - 1 between min and max baselines
- Investigate method outliers



Accessibility: Exploring benchmarking results on the Open Problems website

https://openproblems.bio





Timeline - First release of each task[1]

Date	Repository	Version	Task
2020-07	OpenProblems	v0.1	Label Projection
2020-07	OpenProblems	v0.1	Match Modality
2021-02	OpenProblems	v0.3.1	Regulatory Effect Prediction
2021-02	OpenProblems	v0.3.1	Denoising
2021-02	OpenProblems	v0.3.1	Dimensionality Reduction
2021-07	NeurIPS 2021	v0.1	Predict Modality
2021-07	NeurIPS 2021	v0.1	Joint Embedding
2022-05	OpenProblems	v0.3.4	Batch Integration
2022-08	OpenProblems	v0.5.0	Spatial Decomposition
2022-10	OpenProblems	v0.5.7	Cell-Cell Communication
2024-06	NeurIPS 2023 Perturbation Prediction	v1.0.0	Perturbation Prediction
2024-09	OpenProblems	v2.0.0	Spatially Variable Genes
Ongoing	Spatial Tx Simulators		Spatial Tx Simulators
Ongoing	iST Preprocessing		iST Preprocessing
Ongoing	GRN Inference		GRN Inference

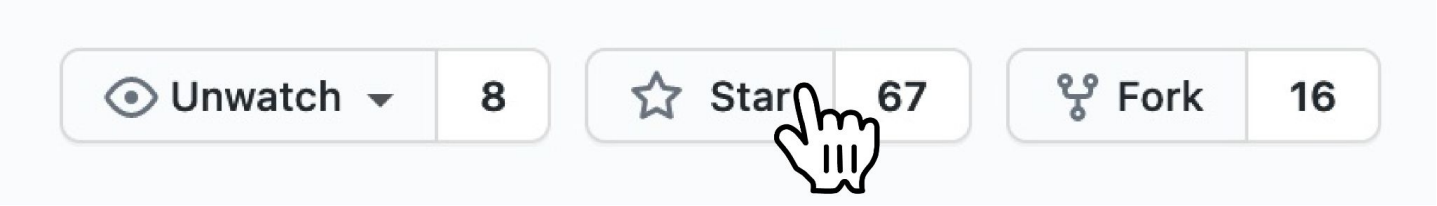
[1] This completely ignores all other improvements made to tasks and the platform that were made along the way!

A deep dive into OpenProblems

Contributing to OpenProblems

- Visit github.com/openproblems-bio/openproblems and hit the "Star" icon
- Adding datasets
- Adding methods
- Adding metrics
- Create a new benchmark from scratch
- Extending the documentation

Your first contribution? 🤦



Want to create a reusable, reproducible, computational benchmark?

Let's get started!

... Hold on a minute! 🤔

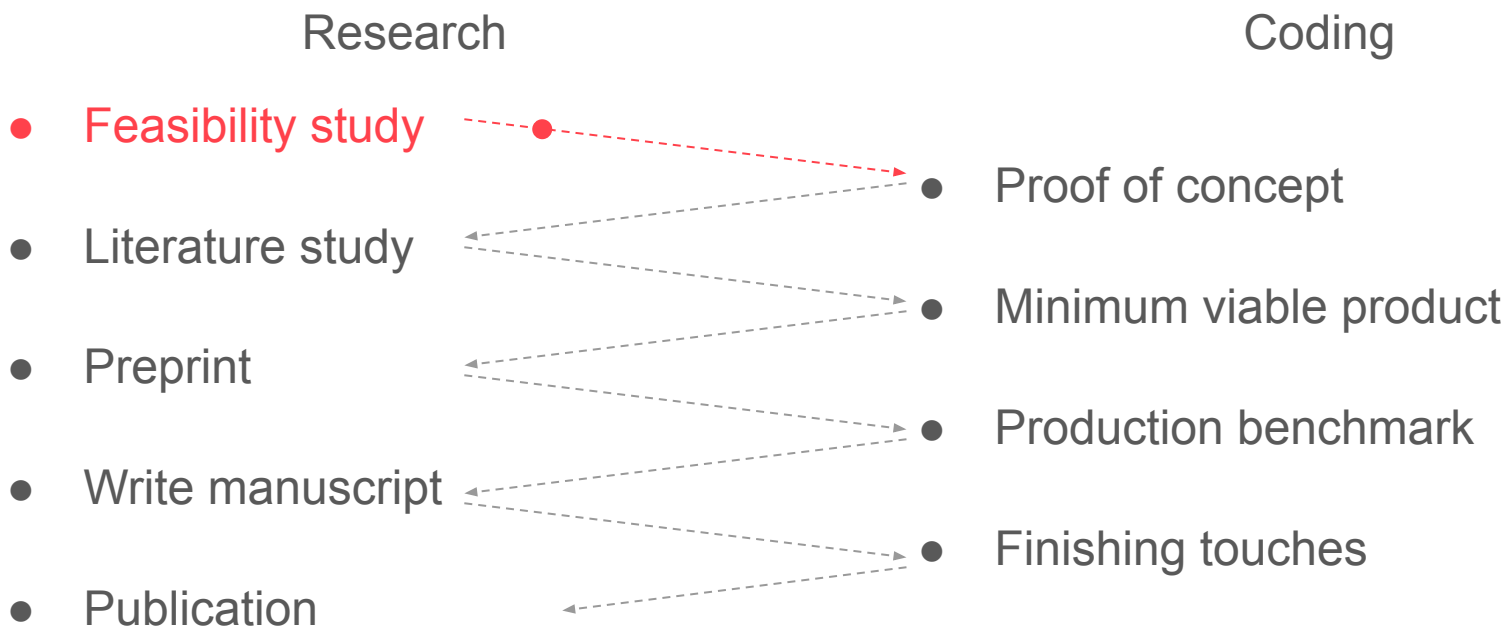
Creating a high-quality benchmark...

- Feasibility study
- Proof of concept
- Literature study
- Minimum viable product
- Preprint
- Production benchmark
- Write manuscript
- Finishing touches
- Publication

... is just like playing Pong

Research	Coding
Feasibility study	
Proof of concept	Proof of concept
Literature study	
Minimum viable product	Minimum viable product
Preprint	
Production benchmark	Production benchmark
Write manuscript	
Finishing touches	Finishing touches
Publication	

Going back and forth is essential for gaining new insights after each step


Research
Coding
Feasibility study
Literature study
Preprint
Write manuscript
Publication
Proof of concept
Minimum viable product
Production benchmark
Finishing touches

Feasibility study

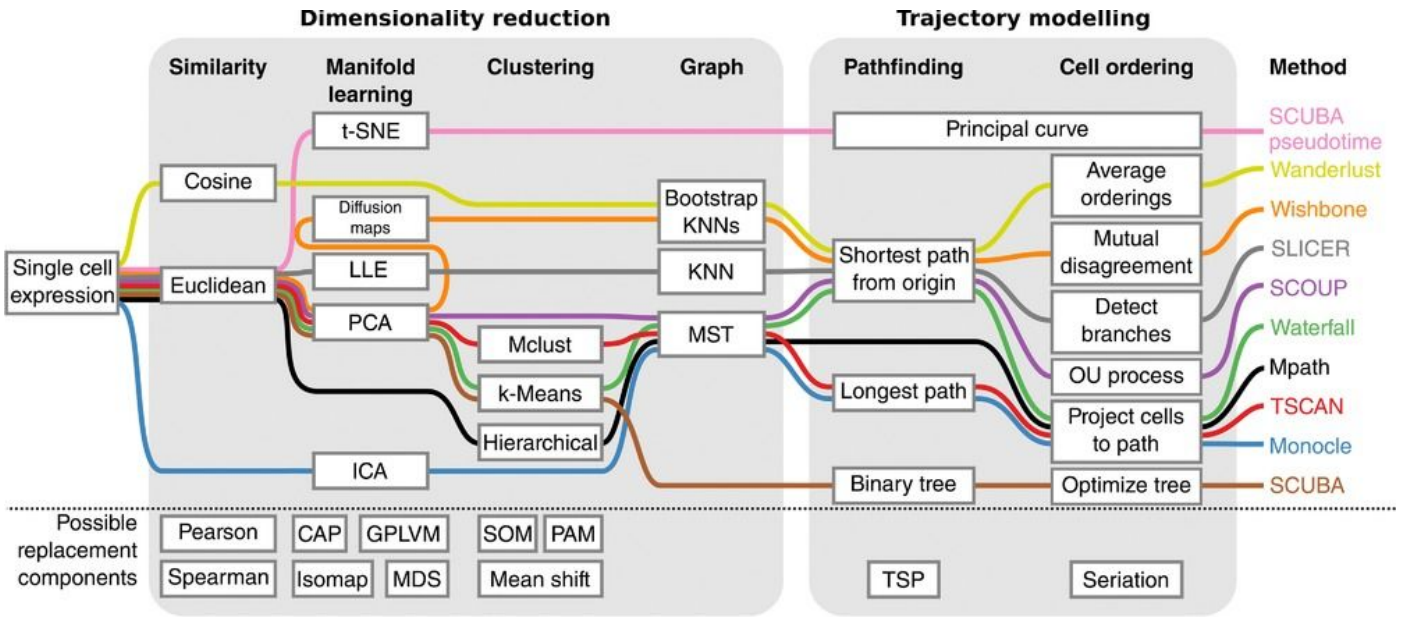
- Formulate the problem you're investigating
- Find datasets (explore the data)
- Run a few computational methods (follow their tutorials)
- In their manuscripts, pay attention to:
  - which research question they are trying to answer
  - which data structures they use
  - with which methods they compare themselves with,
  - on which datasets,
  - using which metrics

Use-case: Trajectory inference

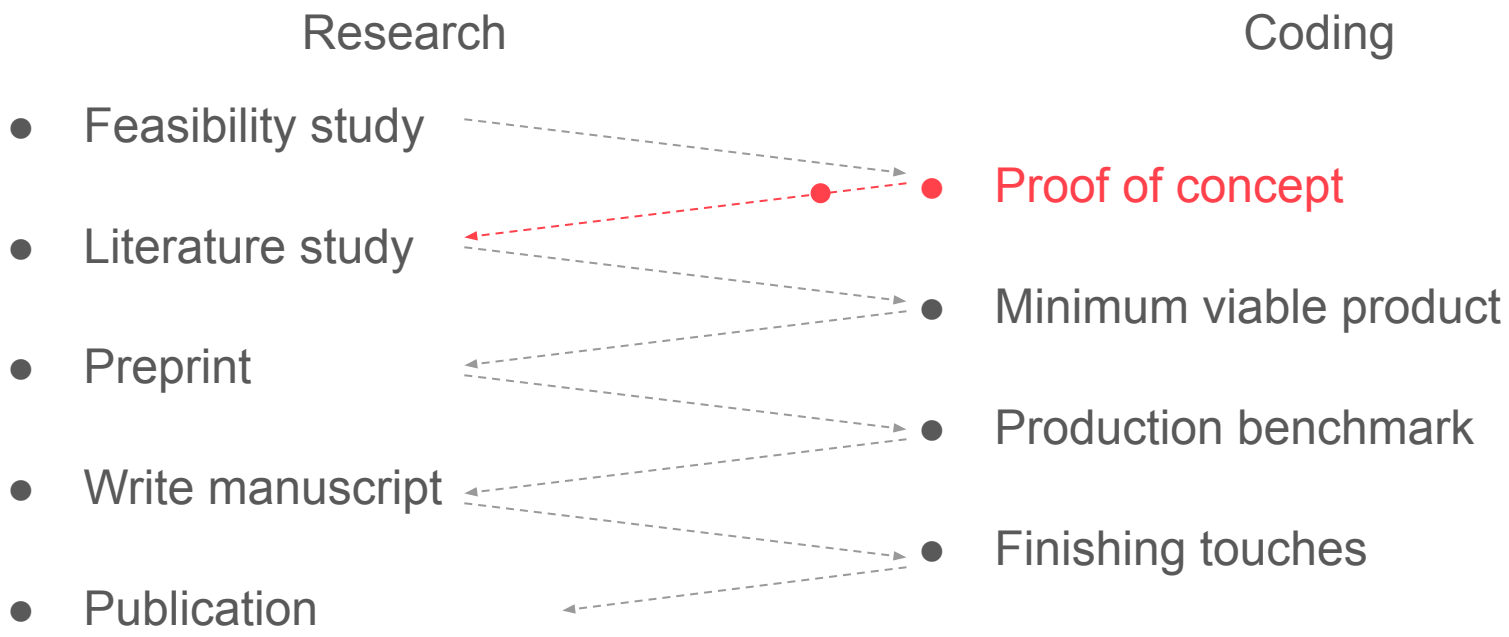
method_id	method_components	method_directionality_infer
angle	any multiple dimensionality reductions {PCA | MDS | tSNE | ICA | LLE} + Arc-tangent	
paga	kNN + louvain clustering + connectivity test { edge statistics | random walk } + diffusion pseudo time	
paga_tree	kNN + louvain clustering + connectivity test { edge statistics | random walk } + diffusion pseudo time + MST	
comp1	any multiple dimensionality reductions {PCA | MDS | tSNE | ICA | LLE}	
celltree_gibbs	LDA (gibbs) + MST (on LDA topics)	
celltree_maptpx	LDA (maptpx) + MST (on LDA topics)	
celltree_vem	LDA (VEM) + MST (on LDA topics)	
dpt	diffusion map + computation of transition matrix + transition matrix to root cell + branch assignment using correlated and anti-correlated distances	root
embeddr	laplacian eigenmap + principal curves	
gpfates	GPLVM + dimensionality reduction from GPLVM + pseudotime inference from GPLVM + OMGP	
mfa	Bayesian hierarchical mixture of factor analyzers + Gibbs sampling	
monocle_ddrtree	any dimensionality reduction + (fuzzy) k-means + MST + move cells towards centers + iteration until convergence + cell projection to tree backbone	
monocle_ica	PCA + ICA + polygonal reconstruction={MST + longest shortest path (from start) + PQ tree}	root
mpath	find landmarks/waypoints + weighted neighbourhood network by number of cells + MST + MBOR ordering of cells = {hierarchical clustering}	
ouija	model expression by combining a mapping function ({switch | transient}) and pseudotime, both as latent variables	
ouijaflow	model expression by combining a mapping function ({switch | transient}) and pseudotime, both as latent variables	
periodpc	PCA + principal curves	
phenopath	latent variable model + mean-field co-ordinate ascent variational inference	
pseudogp	any multiple dimensionality reductions {PCA | MDS | tSNE | ICA | LLE} + GPLVM	

How it started

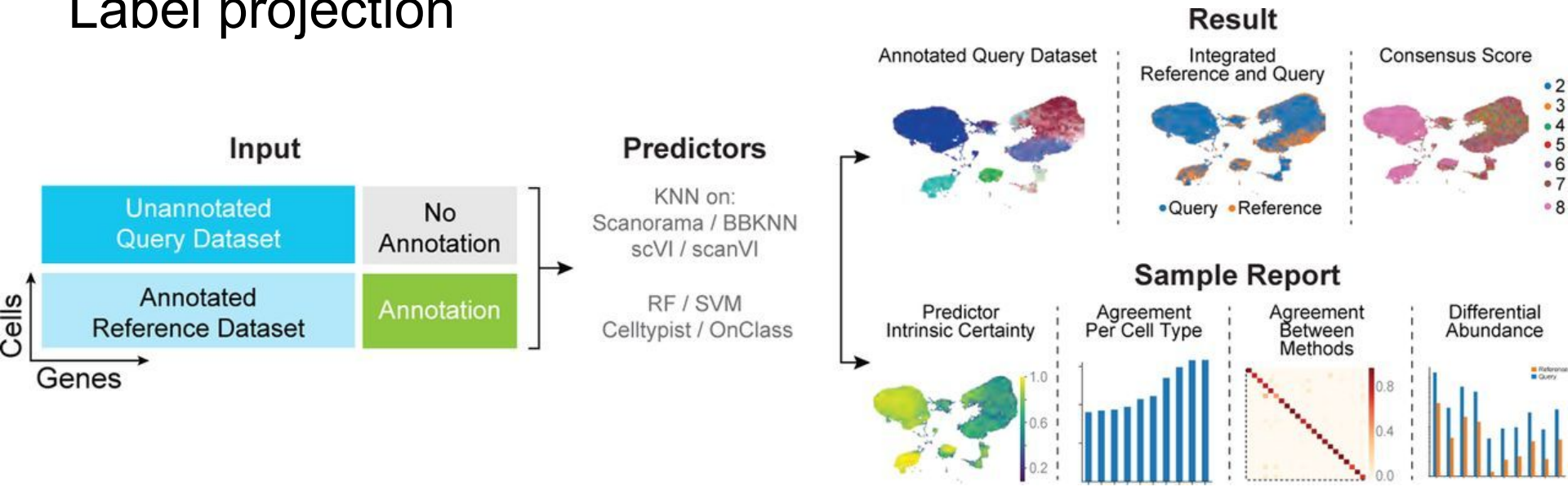
Literature study on trajectory inference methods



Cannoodt, R., Saelens, W. and Saeys, Y., 2016. Computational methods for trajectory inference from single-cell transcriptomics. European journal of immunology, 46(11), pp.2496-2506. doi: 10.1002/eji.201646347


Research
Coding
Feasibility study
Literature study
Preprint
Write manuscript
Publication
Proof of concept
Minimum viable product
Production benchmark
Finishing touches

Label projection



Consensus prediction of cell type labels with popV

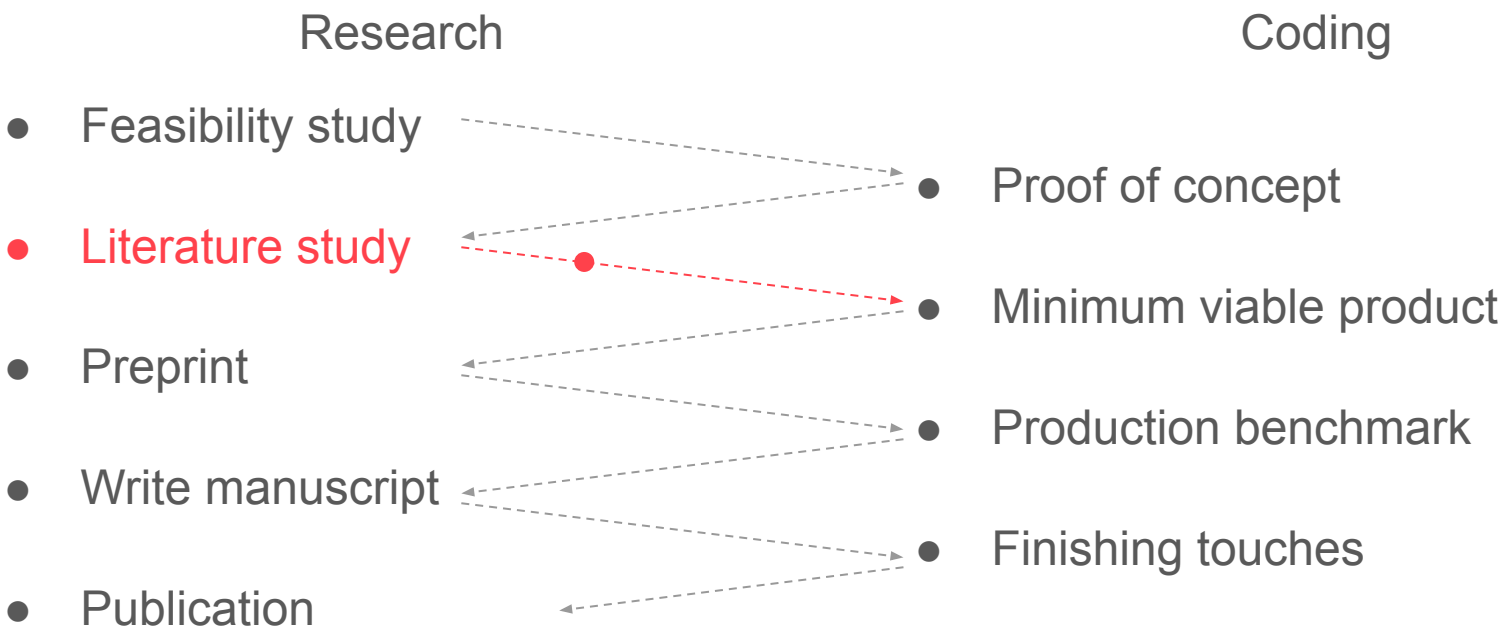
Can Ergen, Galen Xing, Chenling Xu, Michael Jayasuriya, Erin McGeever, Angela Oliveira Pisco, Aaron Streets, Nir Yosef

doi: https://doi.org/10.1101/2023.08.18.553912

POC script

```
import anndata as ad

# Fetch dataset from somewhere.
#   -> This is downloaded from
's3://openproblems-data/resources_test/common/cxg_mouse_pancreas_atlas/dataset.h5ad'
adata = ad.read_h5ad("resources_test/common/cxg_mouse_pancreas_atlas/dataset.h5ad")

adata
# AnnData object with n_obs × n_vars = 600 × 1500
#     obs: 'dataset_id', 'assay', 'assay_ontology_term_id', 'batch', 'cell_type', ...
#     var: 'feature_id', 'feature_name', 'feature_length', 'n_cells', 'hvg', 'hvg_score'
#     uns: 'dataset_description', 'dataset_id', 'dataset_name', 'dataset_organism', 'dataset_reference',
...
#     obsm: 'X_pca'
#     layers: 'counts', 'normalized'

# Process dataset: Divide the dataset into train and test sets.
test_batch = adata.obs["batch"].to_list()[0]
train_adata = adata[adata.obs["batch"] != test_batch].copy()
test_adata = adata[adata.obs["batch"] == test_batch].copy()

# Method: Apply logistic regression
import sklearn.linear_model
classifier = sklearn.linear_model.LogisticRegression()
classifier.fit(train_adata.obsm["X_pca"], train_adata.obs["cell_type"].astype(str))
obs_label_pred = classifier.predict(test_adata.obsm["X_pca"])

# Metric: Compute accuracy
import numpy as np
accuracy = np.mean(test_adata.obs["cell_type"] == obs_label_pred)
accuracy # 0.9478672985781991
```

Gist


Research
Coding
Feasibility study
Literature study
Preprint
Write manuscript
Publication
Proof of concept
Minimum viable product
Production benchmark
Finishing touches

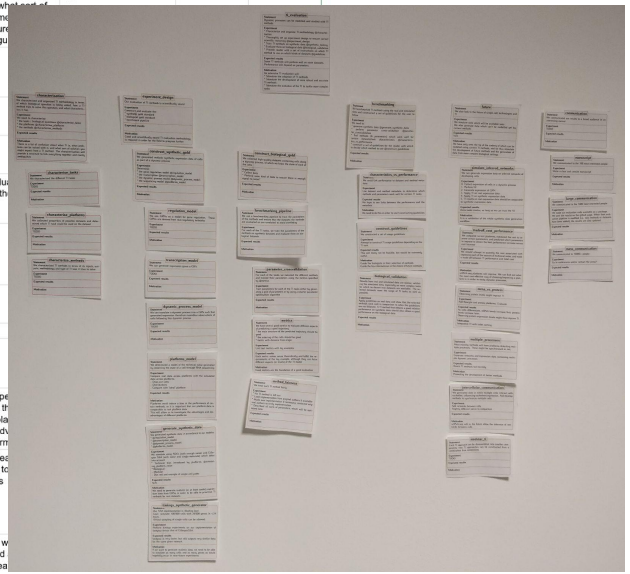
List of hypotheses & tasks

Google sheet:
[Hypotheses and tasks]

Hypotheses & tasks

File Edit View Insert Format Data Tools Extensions Help

experiment_id	order	scien	reader	statement	experiment_method	expected_results	motivation	parent_id	order_sub	color
ti_evaluation	1	10	10	Dynamic processes can be modelled and studied with TI methods	* Characterise and organise TI methodology @characterisation * Thoroughly set up experiment design to ensure correct scientific reasoning @experiment_design * Train TI methods on synthetic data @synthetic_testing * Evaluate them on biological data @biological_validation * Provide reader with a set of instructions on which TI method to use on which kinds of datasets @guidelines	Some TI methods will perform well on some datasets. Performance will depend on parameters.	An extensive TI evaluation will: * Stimulate the adoption of TI methods * Stimulate the development of more robust and accurate TI methods * Stimulate the evolution of the TI to tackle more complex tasks	NA	1	#666666
characterisation	1.1	10	10	We characterised and organised TI methodology in terms of which biological question is being asked, how a TI method tries to solve this question, and what characteristics it has	We need to characterise * the tasks / biological questions @characterise_tasks * the platforms @characterise_platforms * the methods @characterise_methods		There is a lot of confusion about what TI is, what problems can be solved with it, and whe solution you might expect from a TI me characterisation will provide a structure everything together and clarify ambigu	ti_evaluation	1.1	#3498DB
characterise_tasks	1.1.1	10	10	We collected properties of possible datasets and determined which TI task could be used on the dataset	* What trajectory types can or should be discovered by certain methods? * What do these trajectory types have in common, what are the general types?					
characterise_platforms	1.1.2	8	8	We assessed which technologies are used nowadays in single-cell RNA seq	* Extract parameters (library size, cell number, dropout levels, ...) for every platform using reference expression datasets	Platforms vary wildly in their parameters, and these differences will propagate in the evaluation outcome				
characterise_methods	1.1.3	10	10	We characterised TI methods in terms of its inputs, outputs, methodology and type of TI task it tries to solve	TODO					
experiment_design	1.2	10	9	Our evaluation of TI methods is scientifically sound	Construct and evaluate the: * synthetic gold standard * biological gold standard * benchmark pipeline		Good and scientifically sound TI evalua methodology is required in order for th progress further			
construct_synthetic_gold	1.2.1	10	10	We generated realistic synthetic expression data of cells as part of a dynamic process	Determine: * the gene regulation model @regulation_model * the transcription @transcription_model * the dynamic process model @dynamic_process_model * the sequencing model @platforms_model					
regulation_model	1.2.1.1	10	8	We use GRNs as a model for gene regulation. These GRNs are derived from real regulatory networks	TODO					
transcription_model	1.2.1.2	10	8	We can generate expression given a GRN	TODO					
dynamic_process_model	1.2.1.3	10	10	We can translate a dynamic process into a GRN such that generated expression therefrom resembles observations of cells following this dynamic process	TODO					
platforms_model	1.2.1.4	10	6	We determined a model of the technical noise generated by observing the state of a cell through RNA sequencing	Compare real data across platforms with the simulated data across platforms - Drop-out rates - Distributions - Compare with "ideal" platform		Platforms could induce a bias in the pe of certain methods, so it is important th platform data is comparable to real pla This will allow us to investigate the adv and disadvantages of different platform			
generate_synthetic_data	1.2.1.5	10	8	We generated synthetic data in accordance to our models: * @regulation_model * @transcription_model * @dynamic_process_model * @platforms_model	We simulate using SDEs (with enough noise) and Gillespie SSA (with noise and single-molecule) which takes into account * Technical bias introduced by platforms @simulating_platform_noise * Biological - Modular - Dim red and example of single-cell paths	N/A	We need to generate realistic (or at lea expression data from GRNs in order to prioritise TI methods for real datasets			
timings_synthetic_generator	1.2.1.6	6	3	Our SSA implementation is /blazing fast/. Goal: simulate 100'000 cells with 20'000 genes in <24 hours.	Perform timings experiments on our implementation of fastgssa versus that of GillespieSSA.	fastgssa is very faster, but still outputs very similar data for the same given network	If we want to generate realistic data, w be able to simulate as many cells and genes as would hopefully occur in nea			



Review of manuscript and software

Google sheet:
[Overview of trajectory inference methods for dynverse]

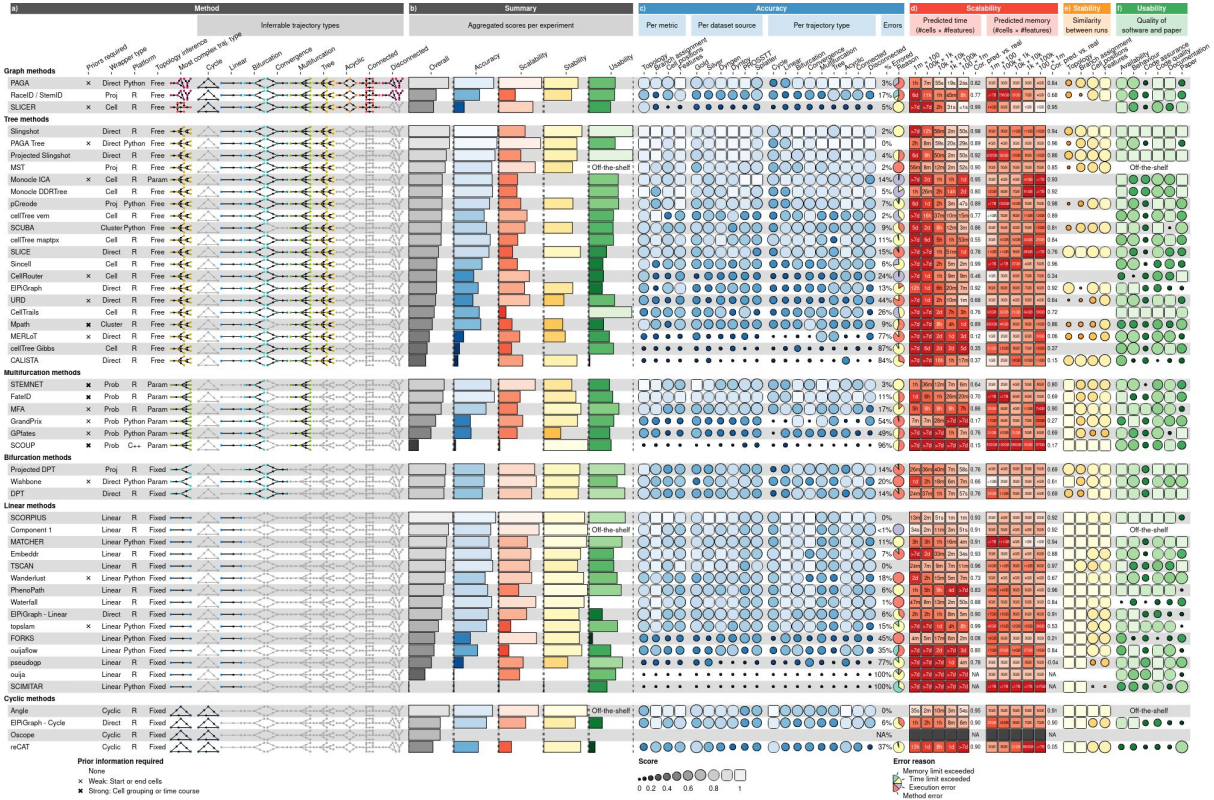
Overview of trajectory inference methods for dynverse

File Edit View Insert Format Data Tools Extensions Help Forms

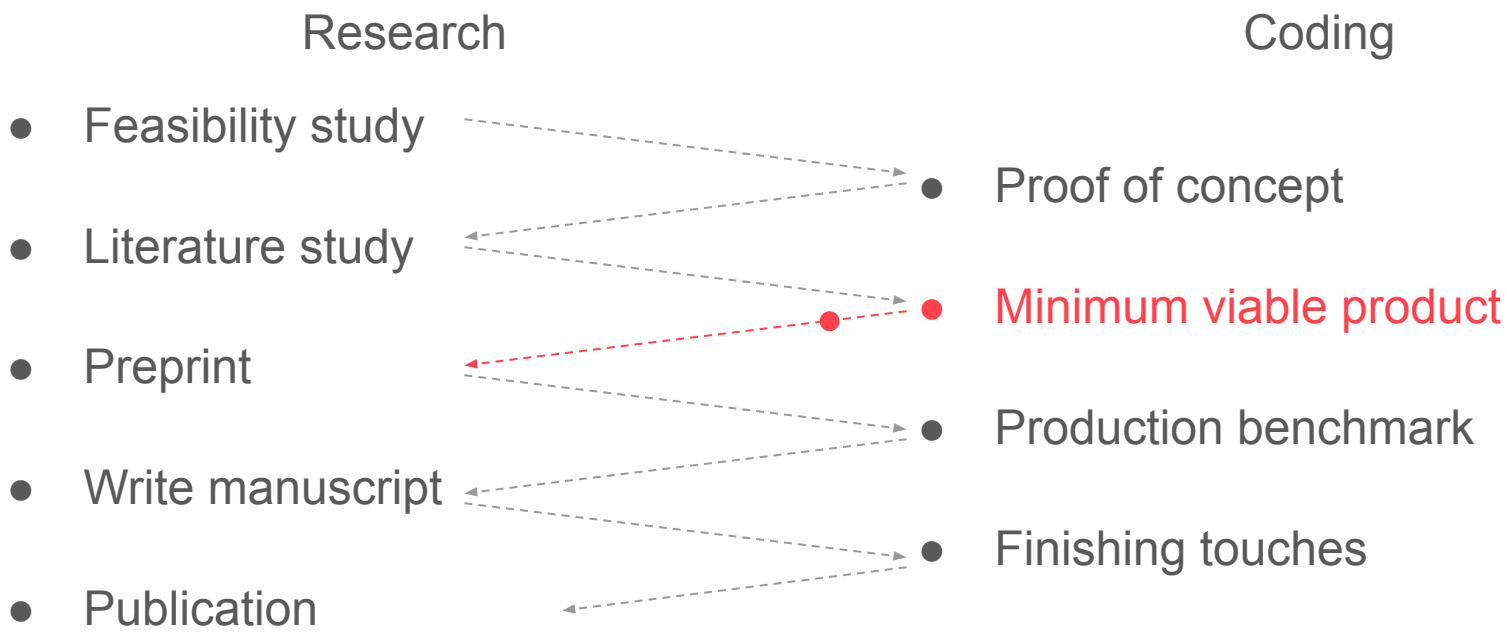
M19 | Plotting functions are provided for the final and/or intermediate results

	A	H	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB
1	name	aspect	item	calista	cellrouter	celltrails	celltree	dpt	elpigraph	embeddr	fateid	forks	gpfates	grandprix	matcher	meriot	mfa	monocle
2			Method's code is freely available	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	Open source	Is the code freely available?	The code can be run on a freely available platform	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4	Version contro	Is the code version controlled?	The code is available on a public version controlled repository, such as Github	1	1 Githu	1	No repo	1 Gitl	1 Github	1 Github	1 Githu	1 Githu	1 Githu	1 Githu	1 Gitl	1 Github	1 Gitl	1 Github
5			The code is provided as a "package", exposing functionality through functions or shell commands		No pac	1 R pac	1 R pac	1 R p	1 R pack	1 R pac	1 R pac	No pac	1 pytho	1 pytho	1 pytl	1 R pack	1 R p	1 R pack
6	Packaging	Is the code contained into an easy to install package? Is it discoverable?	The code can be easily installed through a repository such as CRAN, Bioconductor, PyPI, CPAN, debian packages, ...		Not ava	1 Bioco	1 Bioco	1 Bio	Not avail	Not avai	1 CRAN	Not ava	Not on	Not on	1 Pyf	Not avail	1 Bio	1 Biocon
7			Dependencies are clearly stated in the tutorial or in the code	1	1 In "In	1 R pac	1 R pac	1 R p	1 R pack	1 R pac	1 R pac	1 In the	1 Depe	1 REAL	1 pytl	1 R pack	1 R p	1 R pack
8	Dependencies	Are dependencies clearly stated and available?	Dependencies are automatically installed		User ne	1 R pac	1 R pac	1 R p	1 R pack	1 R pac	1 R pac	Need to	1 Throu	Need to	1 Thr	R depen	1 R p	1 R pack
9		Is the license of the code clear? Does this license permit academic use?	The code is licensed	1 BSD-	No lice	1 Artisti	1 Artisti	1 GP	1 MIT	1 GPL-3	1 GPL-	1 GPL-	1 MIT	No lice	1 MIT	1GPL-3	1 GP	1 Artistic
10	License		License allows academic use	1		1	1	1	1	1	1	1	1		1	1	1	1
11			The tool can be run using a graphical user interface, either locally or on a web server	1					1									
12	Interface	Does the tool have a graphical user interface?	The tool can be run through the command line or through a programming language	No, the	1	1	1	1	1	1	1	No, the	1	1	1	1	1	1
13			Functions/commands have well															

Which resulted in a comparison of TI methods



Saelens, W., Cannoodt, R., Todorov, H. and Saeys, Y., 2019. A comparison of single-cell trajectory inference methods. *Nature biotechnology*, *37*(5), pp.547-554. doi: 10.1038/s41587-019-0071-9


Research
Coding
Feasibility study
Literature study
Preprint
Write manuscript
Publication
Proof of concept
Minimum viable product
Production benchmark
Finishing touches

Building an MVP with OpenProblems

Building a benchmark: two approaches

A focused researcher

Has a **great idea** for a benchmark

Creates a bunch of notebooks to test hypotheses, collect results, and publish

A supplementary zip file with notebooks for running methods and metrics.

- Messy code
- No reproducibility
- Inaccurate results due to bugs

No reusability

Bioinformatics software engineer

Wants to build a **perfect benchmark**

Spends an enormous amount of effort building a benchmarking framework

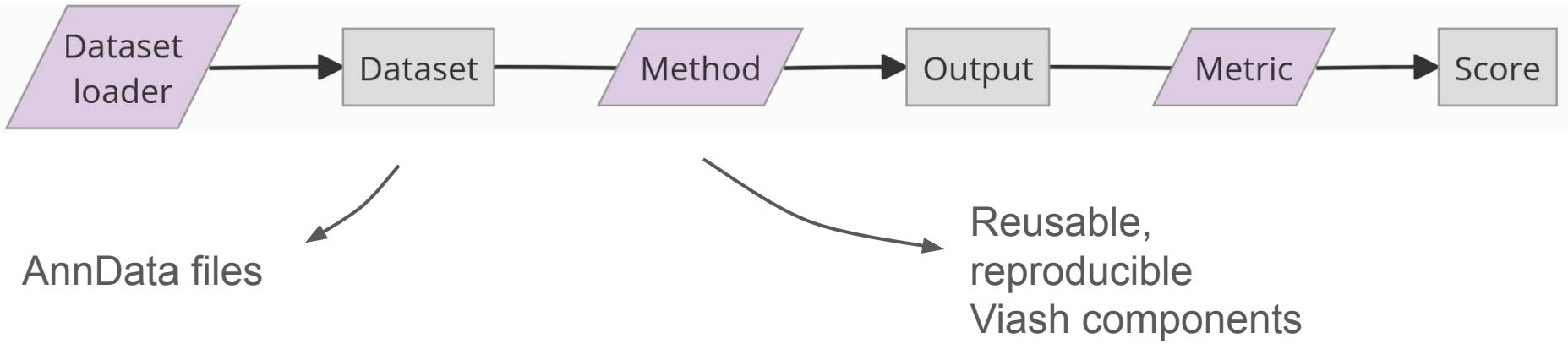
A new benchmarking workflow framework with dataset loaders, methods, metrics, unit tests, Docker containers, CI, CD, Linters, Docs, ...

- Massive effort
- Custom-tailored benchmark workflow
- Does not finish on time, abandons project

No reusability

https://openproblems.bio/documentation/fundamentals/philosophy

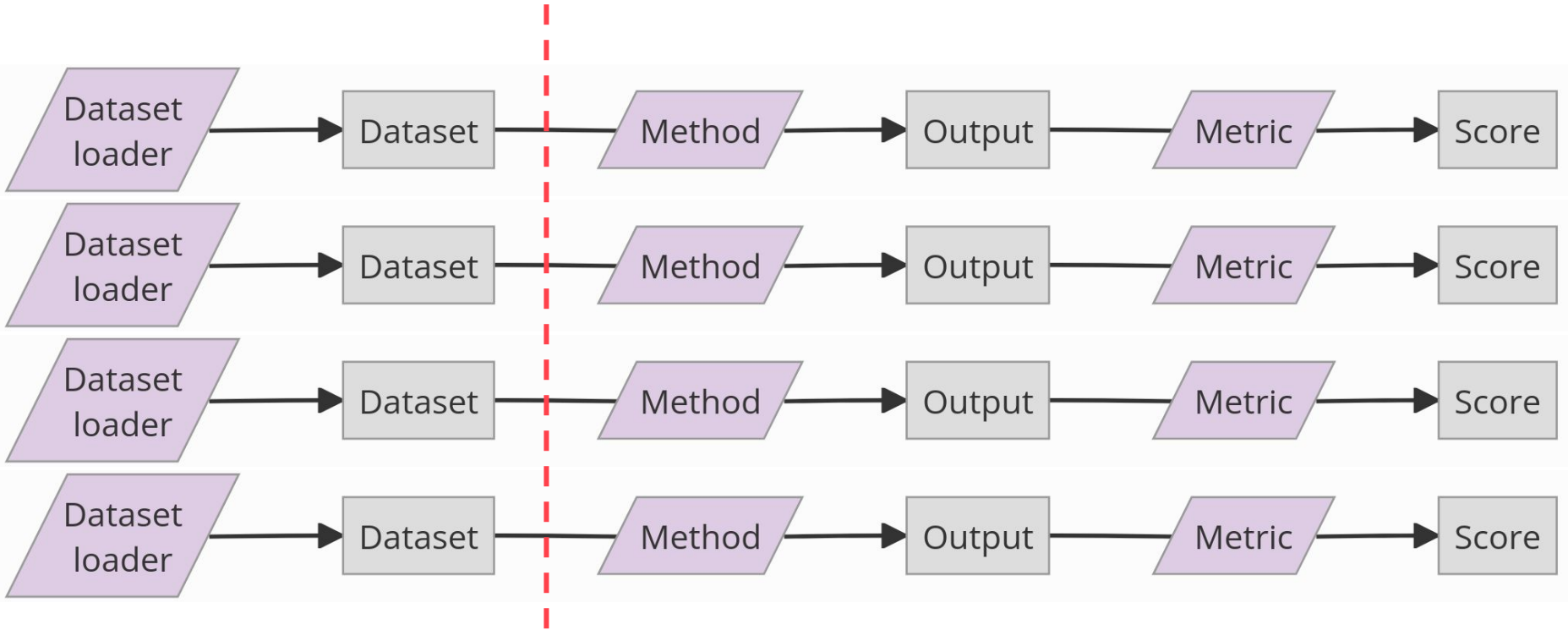
Structure of pretty much any benchmark



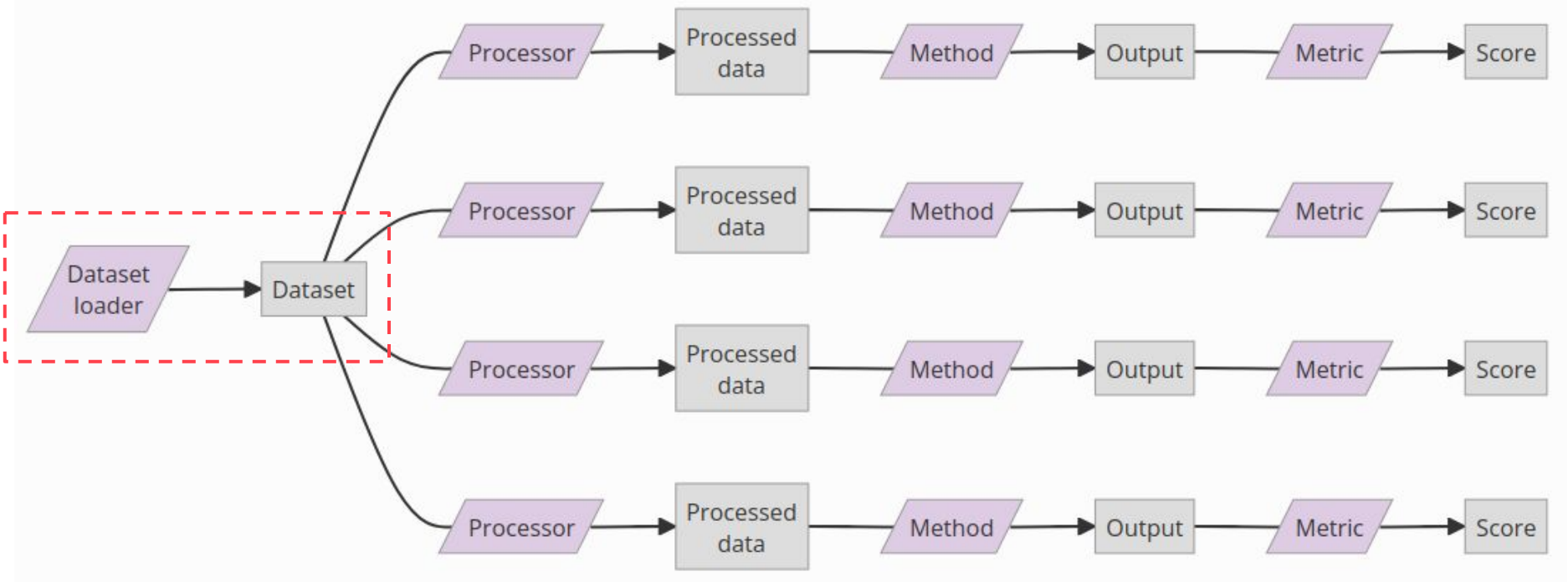
Legend: Grey rectangles are HDF5-backed AnnData (.h5ad) files. Purple rhomboids are Viash components.

https://openproblems.bio/documentation/fundamentals/philosophy

Structure of pretty much any benchmark

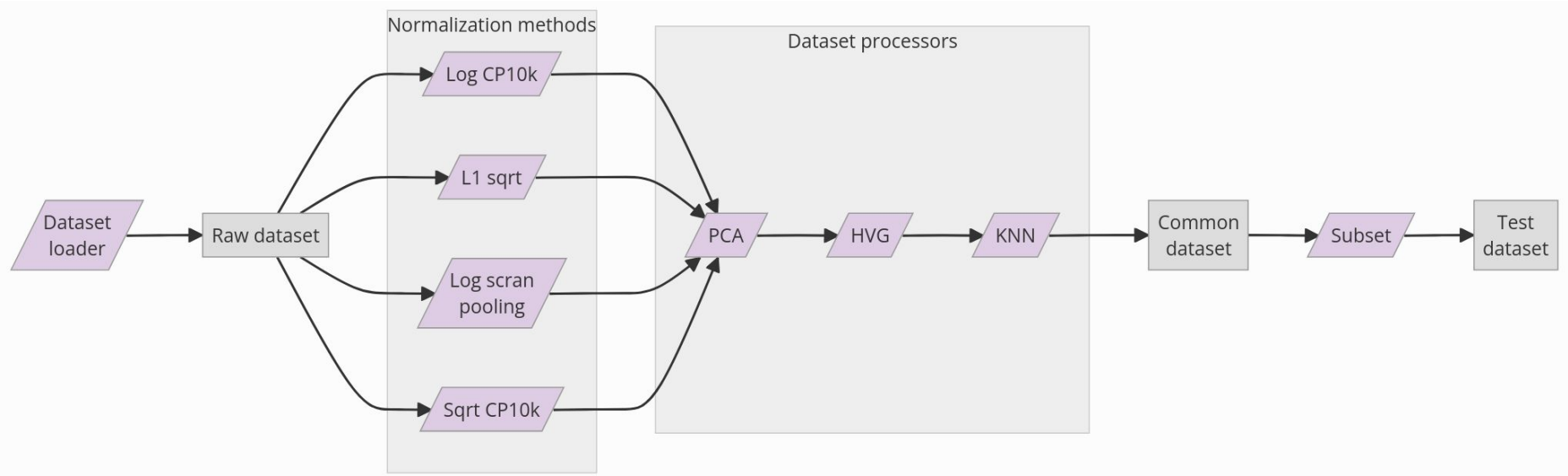


Reusable datasets



https://openproblems.bio/documentation/fundamentals/datasets

Common scRNA-seq dataset processing



Output: a Common dataset in the form of an .h5ad file

Current modalities: scRNA-seq, multi-omics (GEX+ADT, GEX+ATAC), and spatial Tx

https://openproblems.bio/documentation/fundamentals/datasets

All datasets[1] have a corresponding schema file[2]

This "Common dataset H5AD" schema is based on the CELLxGENE schema with some extra metadata.

`cellxgene_census/hlca` is an AnnData object with n_obs × n_vars = 584944 × 28024 with slots:

- obs: `soma_joinid, dataset_id, assay, assay_ontology_term_id, cell_type, cell_type_ontology_term_id, development_stage, development_stage_ontology_term_id, disease, disease_ontology_term_id, donor_id, is_primary_data, self_reported_ethnicity, self_reported_ethnicity_ontology_term_id, sex, sex_ontology_term_id, suspension_type, tissue, tissue_ontology_term_id, tissue_general, tissue_general_ontology_term_id, batch, size_factors`
- var: `soma_joinid, feature_id, feature_name, hvg, hvg_score`
- obsp: `knn_connectivities, knn_distances`
- obsm: `X_pca`
- varm: `pca_loadings`
- layers: `counts, normalized`
- uns: `dataset_description, dataset_id, dataset_name, dataset_organism, dataset_reference, dataset_summary, dataset_url, knn, normalization_id, pca_variance`

[1] h5ad, zarr, YAML, JSON, parquet, csv, tsv

[2] All input/output files for workflows. Data files in intermediary workflow steps might not have a schema associated with them.

File schema

Each input/output file has a specification of which slots are supposed to contain which information.

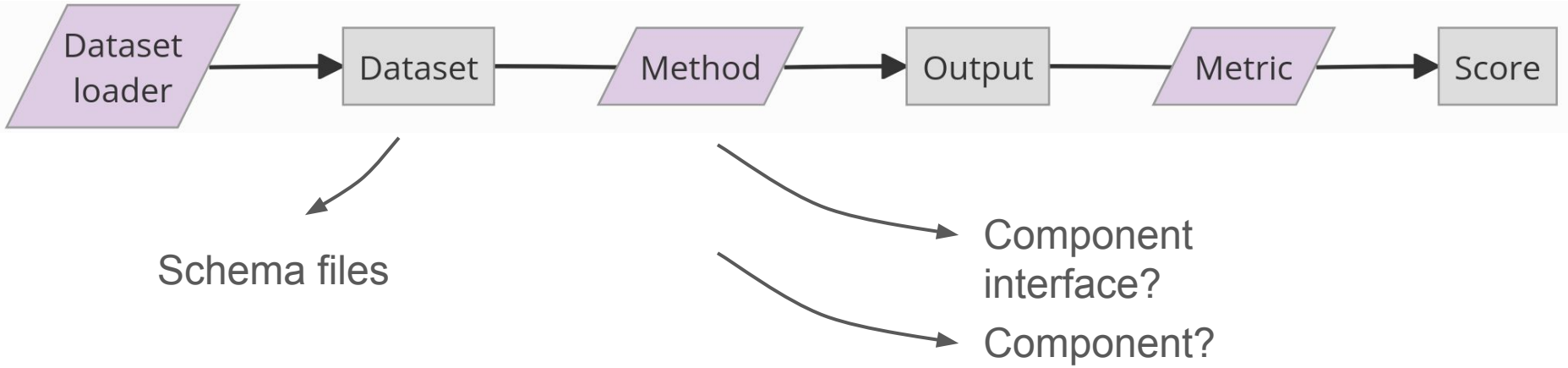
→ Used for documentation and unit testing

Example: link

```yaml
type: file
example: "resources_test/task_label_projection/pancreas/dataset.h5ad"
label: "Training data"
summary: "A dataset containing the train cells and their labels."
info:
 format:
   type: h5ad
   layers:
     - type: integer
       name: counts
       description: Raw counts
       required: true
     - type: double
       name: normalized
       description: Normalized expression values
       required: true
   obs:
     - type: string
       name: cell_type
       description: Ground truth cell type labels
       required: true
   uns:
     - type: string
       name: dataset_id
       description: "A unique identifier for the dataset"
       required: true
     - type: string
       name: normalization_id
       description: "Which normalization was used"
       required: true
```

Dataset file schema: ✓

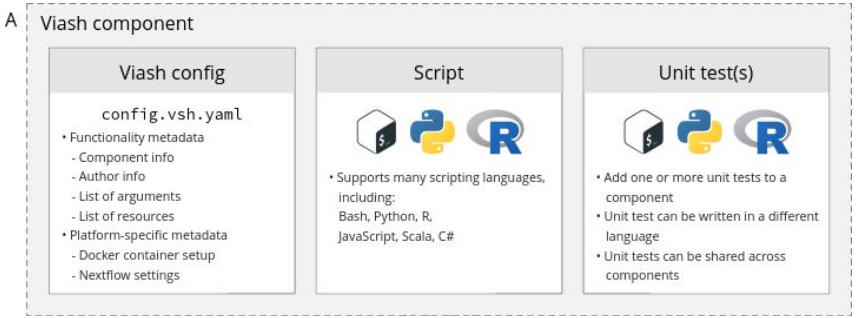
Component interfaces: ?



Legend: Grey rectangles are HDF5-backed AnnData (.h5ad) files. Purple rhomboids are Viash components.

https://openproblems.bio/documentation/fundamentals/components

Viash components (a bit like pandoc, but for scripts)



Example of a Viash component

```yaml
# contents of 'config.vsh.yaml'
# Metadata
name: logistic_regression
namespace: methods
label: Logistic Regression
summary: Logistic Regression with  [...]
description: |
  Logistic Regression estimates parameters [...]
references:
  doi: 123456/123456.1234.789456
links:
  repository: https://github.com/sciki t-lea[...]
  documentation: https://scikit-learn.org/s [...]
# Interface
...
```

```yaml
# contents of 'config.vsh.yaml' (cont)
arguments: # Interface
  - name: "--input_train"
    type: file
    direction: input
    required: true
  - name: "--input_test"
    type: file
    direction: input
    required: true
  - name: "--output"
    type: file
    direction: output
    required: true
```

```python
# script.py:
import anndata as ad
import sklearn.linear_model
import pandas as pd

train_adata = ad.read_h5ad(par["input_train"])
test_adata = ad.read_h5ad(par["input_test"])

# train model
classifier = \
  sklearn.linear_model.LogisticRegression()
classifier.fit(train_adata.obsm["X_pca"],
train_adata.obs["cell_type"].astype(str))

# create output
obs_label_pred = classifier.predict(
 test_adata.obsm["X_pca"]
)

# create output
output = ad.AnnData(
  obs = pd.DataFrame(
    { "pred": obs_label_pred },
    index = test_adata.index
  )
)
output.write_h5ad(par["output"])
```

Component API

Common settings such as input/output arguments, unit tests and test resources.

Example: link

```yaml
namespace: methods
info:
  type: method
  type_info:
    label: Method
    summary: A label projection method.
    description: |
      A label projection method to predict the labels of a new "test"
      dataset based on an annotated "training" dataset.
arguments:
  - name: "--input_train"
    __merge__: file_train.yaml
    direction: input
    required: true
  - name: "--input_test"
    __merge__: file_test.yaml
    direction: input
    required: true
  - name: "--output"
    __merge__: file_prediction.yaml
    direction: output
    required: true
test_resources:
  - # paths to unit tests and test resource
```

Actual method config

The amount of information a contributor needs to provide in a Viash config has been reduced as much as possible.

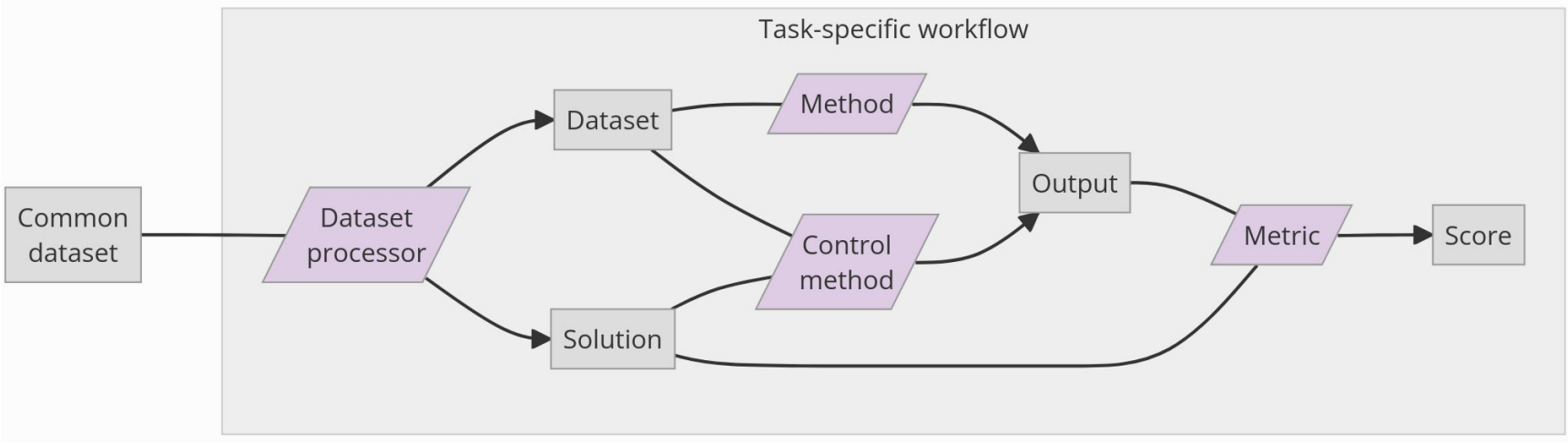
Example: link

Viash docs: viash.io

```yaml
# Common settings such as input/output arguments, unit tests and test resources
__merge__: /src/api/comp_method.yaml
# Metadata
name: logistic_regression
label: Logistic Regression
summary: Logistic Regression with 100-dimensional PCA coordinates
description: |
  Logistic Regression estimates parameters of a logistic function for [...]
references:
  doi: 123456/123456.1234.789456
links:
  repository: https://github.com/scikit-learn/scikit-learn
  documentation: https://scikit-learn.org/stable/[...]
# Code
resources:
  - type: python_script
    path: script.py
# Reproducibility
engines:
  - type: docker
    image: openproblems/base_python:1.0.0
    setup:
      - type: python
        packages: scikit-learn
# Usability
runners:
  - type: executable
  - type: nextflow
    directives:
      label: [midtime, midmem, lowcpu]
```

https://openproblems.bio/documentation/create_task/task_structure

Structure of a typical OpenProblems task benchmark

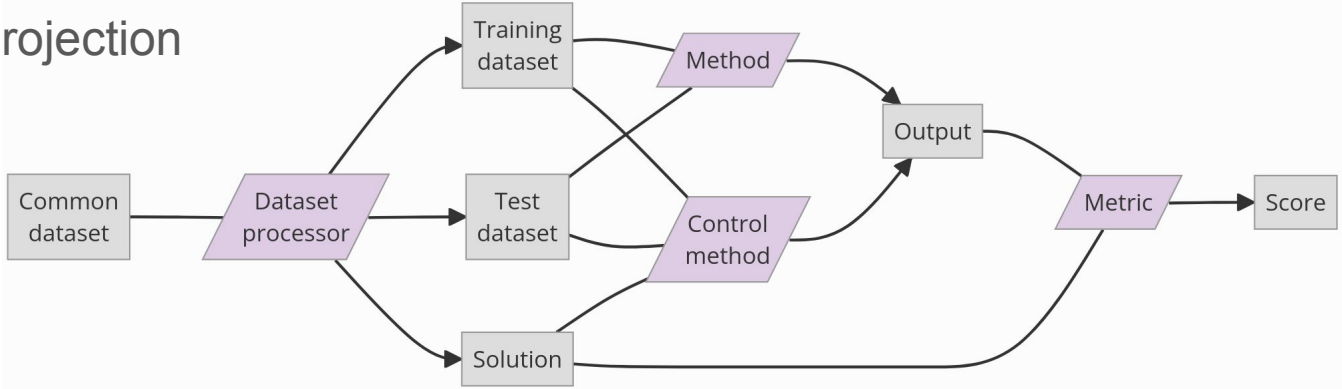


- API: The file schemas and component interfaces for each entity in the above graph
- Data processor: transforms a Common dataset to task-specific data objects
- Method: A computational algorithm to solve the task at hand
- Metric: A performance metric that compares a method output to the solution
- Control methods: Positive or negative control methods which are used for quality control of the task results and scaling of metric values

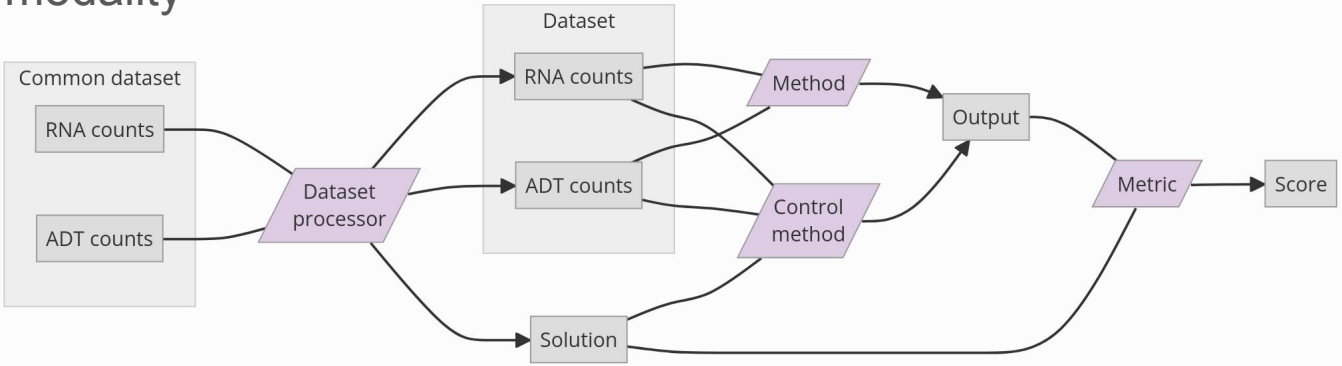
https://openproblems.bio/documentation/create_task/design_api

Topology of a task depends on experimental design

Label projection



Predict modality



https://openproblems.bio/documentation/advanced_topics/safeguards

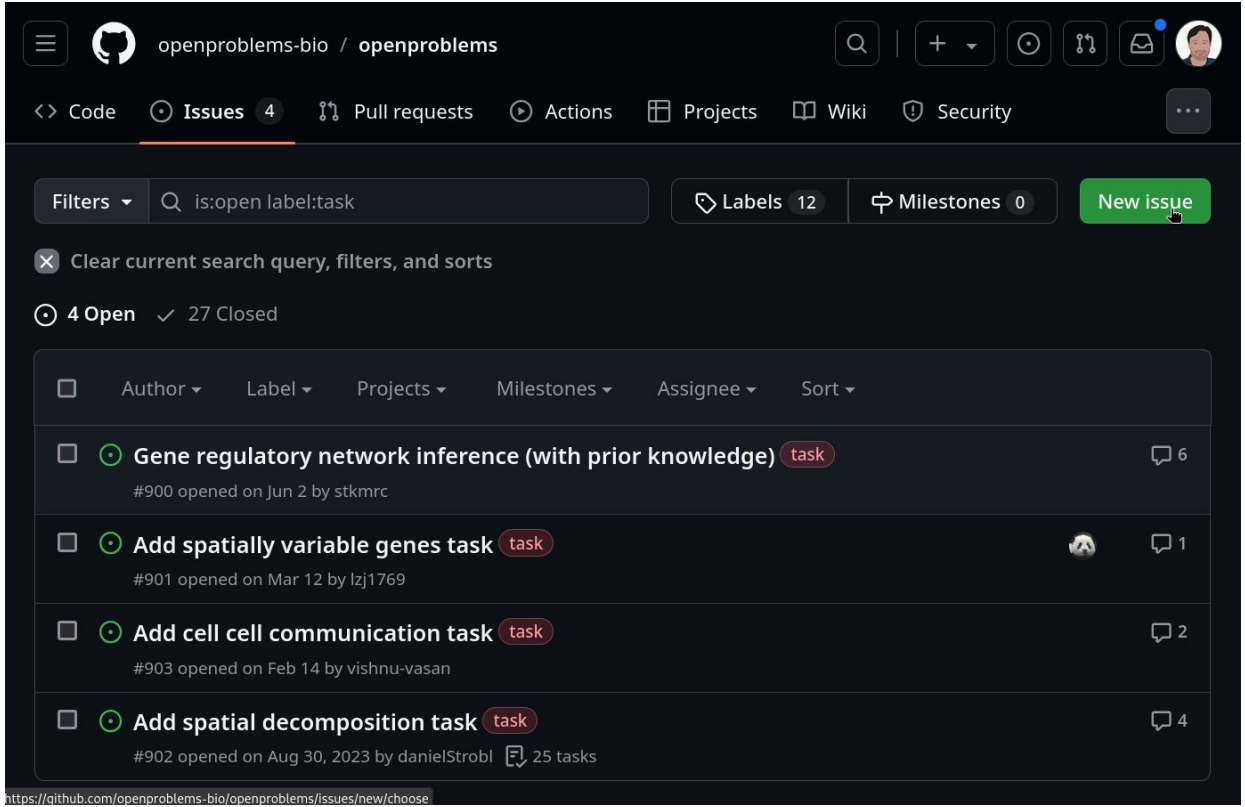
Safeguarding benchmark integrity

- Reuse common datasets: Can't introduce new bugs in preprocessing
- Control methods: Quality control of the benchmarking results
- File schemas: Automatically validate component outputs, detect bugs earlier
- Automated unit testing
- Automated documentation
- CI CI CI
- Versioned releases of components, workflows, and Docker images
- Run anywhere: locally, cloud (AWS, Google, Azure), HPC

Creating a new task

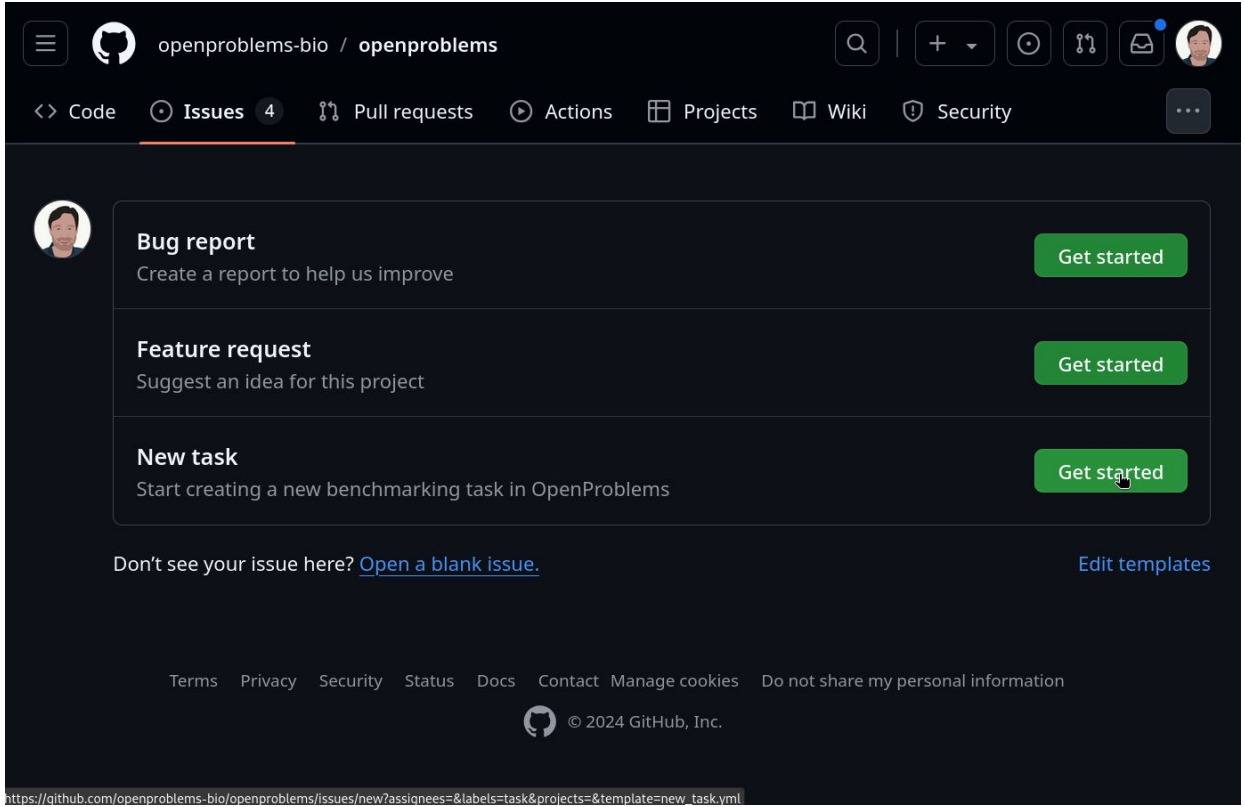
https://openproblems.bio/documentation/create_task/getting_started

Step 1: Check whether a similar task already exists



https://openproblems.bio/documentation/create_task/getting_started

Step 2: Reach out to the OpenProblems team



https://openproblems.bio/documentation/create_task/getting_started

Step 2: Reach out to the OpenProblems team

Issue: New task

Start creating a new benchmarking task in OpenProblems. If this doesn't look right, choose a different type.

Add a title

Title

Thanks for choosing OpenProblems. Please check the OpenProblems tasks to see whether a similar task has already been created. If you haven't already, please review the documentation on how to create a new task.

Task motivation

Explain the motivation behind your proposed task. Describe the biological or computational problem you aim to address and why it's important. Discuss the current state of research in this area and any gaps or challenges that your task could help address. This section should convince readers of the significance and relevance of your task.

(Basically the information from your feasibility study)

Task description

Provide a clear and concise description of your task, detailing the specific problem it

Assignees

No one—assign yourself

Labels

task

Projects

None yet

Milestone

No milestone

Development

Shows branches and pull requests linked to this issue.

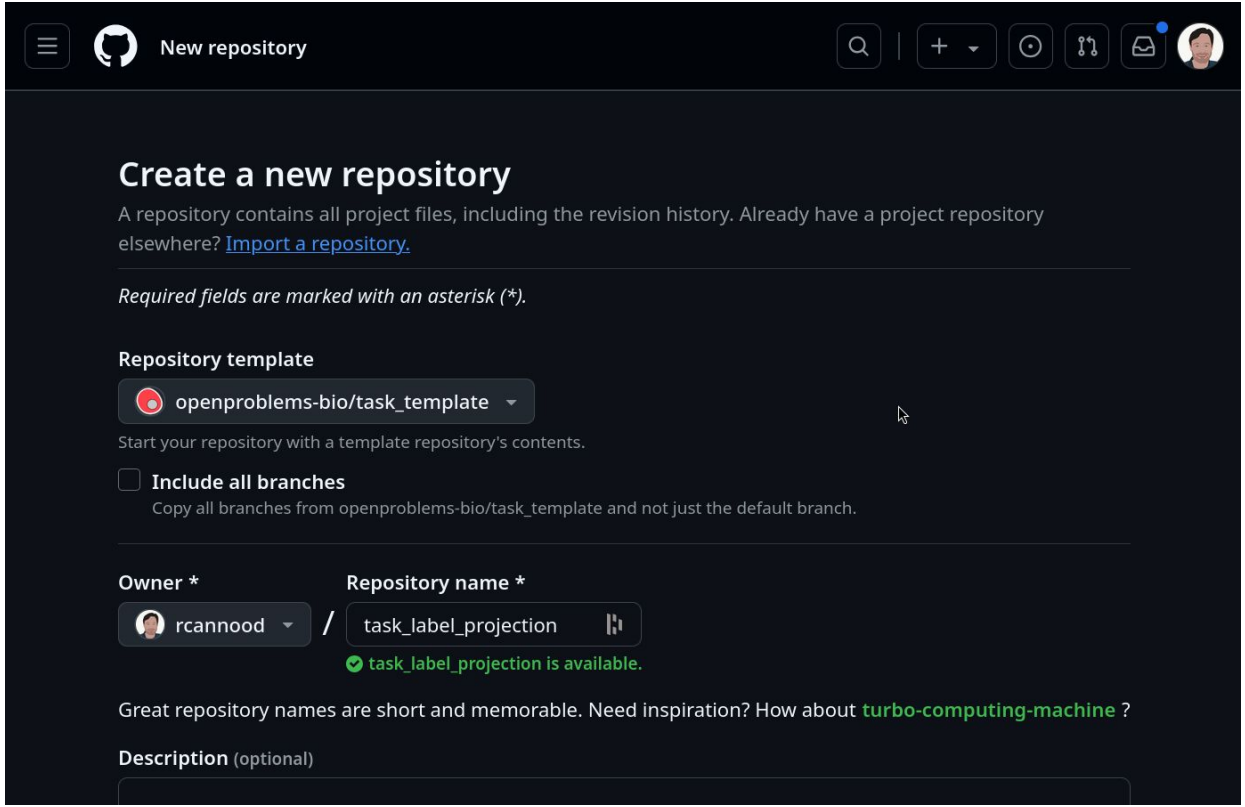
Helpful resources

Contributing
Code of conduct
GitHub Community Guidelines

You're using an issue form, a

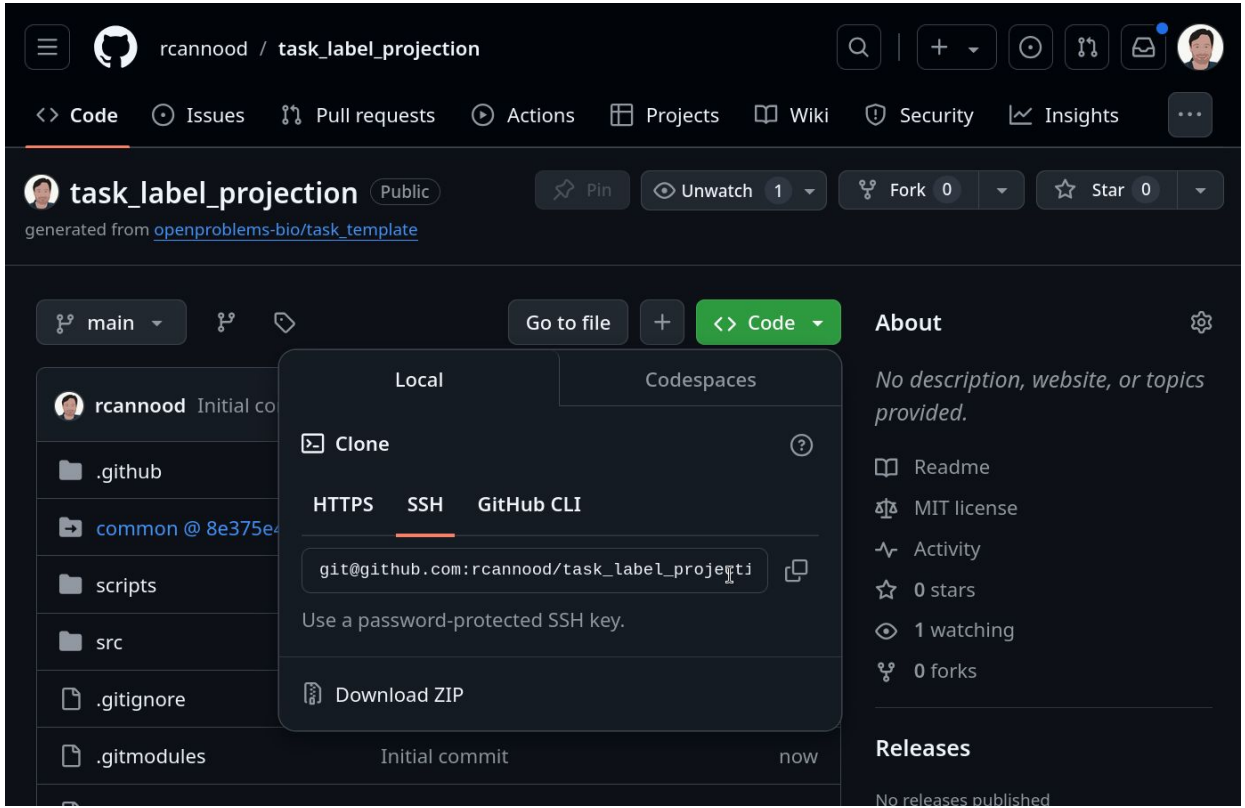
https://openproblems.bio/documentation/create_task/getting_started

Step 3: Create task repository



https://openproblems.bio/documentation/create_task/getting_started

Step 4: Clone the task repository



https://openproblems.bio/documentation/create_task/getting_started

Step 4 & 5: Clone and test the task template

```
# Clone repository:
git clone --recursive git@github.com:rcannood/task_label_projection.git
cd task_label_projection

# Sync test resources
scripts/sync_resources.sh

# Run one component
viash run src/methods/logistic_regression/config.vsh.yaml -- --help

viash run src/methods/logistic_regression/config.vsh.yaml -- \
   --input_train resources_test/task_template/pancreas/train.h5ad \
   --input_test resources_test/task_template/pancreas/test.h5ad \
   --output output.h5ad

# Build all components:
viash ns build --parallel --setup cachedbuild

# Test one component:
viash test src/methods/logistic_regression/config.vsh.yaml

# Run the benchmarking workflow on the test datasets
scripts/run_benchmark/run_test_local.sh
```

https://openproblems.bio/documentation/create_task/task_structure

Step 6: Look at repository structure

```
tree -L 2
├── CHANGELOG.md
├── common                    # git submodule with common resources
├── CONTRIBUTING.md
├── LICENSE
├── main.nf
├── nextflow.config
├── README.md
├── resources_test            # test resources
│   ├── common                # common test resources you get after running sync_resources
│   └── task_template         # task-specific test resources you need to generate
├── scripts
│   ├── create_component      # generator script for creating a new method or metric
│   ├── create_readme.sh      # generate README from AI files
│   ├── create_resources      # scripts to generate test resources
│   ├── project               # build and test scripts
│   ├── run_benchmark         # run the benchmark
│   └── sync_resources.sh     # fetch test resources
├── src
│   ├── api                   # the api yamls
│   ├── control_methods       # control method components
│   ├── data_processors       # dataset processor
│   ├── methods               # method components
│   ├── metrics               # metrics components
│   └── workflows             # viash nextflow workflow
└── _viash.yaml               # project config
```

https://openproblems.bio/documentation/create_task/getting_started

Step 7: Update project config '_viash.yaml'

```
tree -L 2
├── CHANGELOG.md
├── common                    # git submodule with common resources
├── CONTRIBUTING.md
├── LICENSE
├── main.nf
├── nextflow.config
├── README.md
├── resources_test            # test resources
│   ├── common                # common test resources you get after running sync_resources
│   └── task_template         # task-specific test resources you need to generate
├── scripts
│   ├── create_component      # generator script for creating a new method or metric
│   ├── create_readme.sh      # generate README from AI files
│   ├── create_resources      # scripts to generate test resources
│   ├── project               # build and test scripts
│   ├── run_benchmark         # run the benchmark
│   └── sync_resources.sh     # fetch test resources
├── src
│   ├── api                   # the api yamls
│   ├── control_methods       # control method components
│   ├── data_processors       # dataset processor
│   ├── methods               # method components
│   ├── metrics               # metrics components
│   └── workflows             # viash nextflow workflow
└── _viash.yaml               # project config <-------------------------------------------------------------------
```

https://openproblems.bio/documentation/create_task/getting_started

Step 7: Update project config '_viash.yaml'

```yaml
name: task_label_projection

# documentation
label: Label Projection
summary: A one-sentence summary of the purpose and methodology of this benchmark.
description: |
 A multi-line description of the motivation behind this task and its experimental design.
 If you followed the previous steps, you can copy paste this information from 'Step 2'.
references:
 doi: [ 10.1234/123456.7890.123456 ]
keywords: [ scRNA-seq, label projection ]
authors:
 - name: John Doe
license: MIT

# technical information
# ...
```

https://openproblems.bio/documentation/create_task/design_api

Step 8: Create file schemas

```
tree -L 2                ✔
├── CHANGELOG.md
├── common                    # git submodule with common resources
├── CONTRIBUTING.md
├── LICENSE
├── main.nf
├── nextflow.config
├── README.md
├── resources_test            # test resources
│   ├── common                # common test resources you get after running sync_resources
│   └── task_template         # task-specific test resources you need to generate
├── scripts
│   ├── create_component      # generator script for creating a new method or metric
│   ├── create_readme.sh      # generate README from AI files
│   ├── create_resources      # scripts to generate test resources
│   ├── project               # build and test scripts
│   ├── run_benchmark         # run the benchmark
│   └── sync_resources.sh     # fetch test resources
├── src
│   ├── api                   # the api yamls <-------------------------------------------------------------
│   ├── control_methods       # control method components
│   ├── data_processors       # dataset processor
│   ├── methods               # method components
│   ├── metrics               # metrics components
│   └── workflows             # viash nextflow workflow
└── _viash.yaml               # project config
```

https://openproblems.bio/documentation/create_task/design_api

Step 8: Create API

```
tree src/api
├── comp_control_method.yaml
├── comp_data_processor.yaml
├── comp_method.yaml
├── comp_metric.yaml
├── file_common_dataset.yaml
├── file_prediction.yaml
├── file_score.yaml
├── file_solution.yaml
├── file_test_h5ad.yaml
└── file_train_h5ad.yaml

1 directory, 10 files
```

https://openproblems.bio/documentation/create_task/design_api

Step 8: Create API - file schemas (src/api/file_train.yaml)

```yaml
type: file
example: "resources_test/task_label_projection/pancreas/dataset.h5ad"
label: "Training data"
summary: "A dataset containing the train cells and their labels."
info:
 format:
   type: h5ad
   layers:
     - type: integer
       name: counts
       description: Raw counts
       required: true
     - type: double
       name: normalized
       description: Normalized expression values
       required: true
   obs:
     - type: string
       name: cell_type
       description: Ground truth cell type labels
       required: true
   uns:
     - type: string
       name: dataset_id
       description: "A unique identifier for the dataset"
       required: true
     - type: string
       name: normalization_id
       description: "Which normalization was used"
       required: true
```

https://openproblems.bio/documentation/create_task/design_api

Step 8: Create API - component interface (src/api/comp_method.yaml)

```yaml
namespace: methods
info:
  type: method
  type_info:
    label: Method
    summary: A label projection method.
    description: |
      A label projection method to predict the labels of a new "test"
      dataset based on an annotated "training" dataset.
arguments:
  - name: "--input_train"
    __merge__: file_train.yaml
    direction: input
    required: true
  - name: "--input_test"
    __merge__: file_test.yaml
    direction: input
    required: true
  - name: "--output"
    __merge__: file_prediction.yaml
    direction: output
    required: true
test_resources:
  - # paths to unit tests and test resources
```

https://openproblems.bio/documentation/create_task/design_api

Step 9: Render README

`scripts/create_readme.sh`

README MIT license

Label projection

Automated cell type annotation from rich, labeled reference data

Repository: openproblems-bio/task_label_projection

Description

A major challenge for integrating single cell datasets is creating matching cell type annotations for each cell. One of the most common strategies for annotating cell types is referred to as "cluster-then-annotate" whereby cells are aggregated into clusters based on feature similarity and then manually characterized based on differential gene expression or previously identified marker genes. Recently, methods have emerged to build on this strategy and annotate cells using known marker genes. However, these strategies pose a difficulty for integrating atlas-scale datasets as the particular annotations may not match.

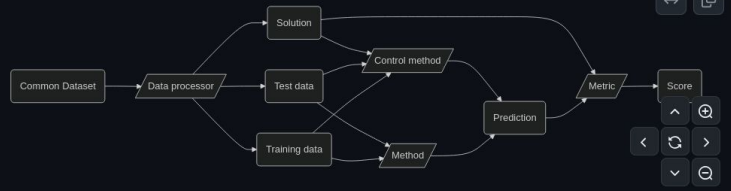
To ensure that the cell type labels in newly generated datasets match existing reference datasets, some methods align cells to a previously annotated reference dataset and then *project* labels from the reference to the new dataset.

Here, we compare methods for annotation based on a reference dataset. The datasets consist of two or more samples of single cell profiles that have been manually annotated with matching labels. These datasets are then split into training and test batches, and the task of each method is to train a cell type classifer on the training set and project those labels onto the test set.

Authors & contributors

name	roles
Nikolay Markov	author, maintainer
Scott Gigante	author
Robrecht Cannoodt	author

API



File format: Common Dataset

A subset of the common dataset.

Example file: `resources_test/common/pancreas/dataset.h5ad`

Format:

```
AnnData object
 obs: 'cell_type', 'batch'
 var: 'hvg', 'hvg_score'
 obsm: 'X_pca'
 layers: 'counts', 'normalized'
 uns: 'dataset_id', 'dataset_name', 'dataset_url', 'dataset_reference', 'dataset_summary', 'dat
```

Data structure:

Slot	Type	Description
`obs["cell_type"]`	`string`	Cell type information.
`obs["batch"]`	`string`	Batch information.
`var["hvg"]`	`boolean`	Whether or not the feature is considered to be a 'highly variable gene'.

https://openproblems.bio/documentation/create_task/create_components

Step 10: Create components

```
tree -L 2                ✔
├── CHANGELOG.md
├── common                    # git submodule with common resources
├── CONTRIBUTING.md
├── LICENSE
├── main.nf
├── nextflow.config
├── README.md
├── resources_test            # test resources
│   ├── common                # common test resources you get after running sync_resources
│   └── task_template         # task-specific test resources you need to generate
├── scripts
│   ├── create_component      # generator script for creating a new method or metric
│   ├── create_readme.sh      # generate README from AI files
│   ├── create_resources      # scripts to generate test resources
│   ├── project               # build and test scripts
│   ├── run_benchmark         # run the benchmark
│   └── sync_resources.sh     # fetch test resources
├── src
│   ├── api                   # the api yamls
│   ├── control_methods       # control method components <-------------------------------------------------------
│   ├── data_processors       # dataset processor <-------------------------------------------------------------
│   ├── methods               # method components <-------------------------------------------------------------
│   ├── metrics               # metrics components <------------------------------------------------------------
│   └── workflows             # viash nextflow workflow
└── _viash.yaml               # project config
```

https://openproblems.bio/documentation/create_task/create_components

Step 10: Create components

```
tree -L 2                ✔
├── CHANGELOG.md
├── common
├── CONTRIBUTING.md
├── LICENSE
├── main.nf
├── nextflow.config
├── README.md
├── resources_test
│   ├── common
│   └── task_template
├── scripts
│   ├── create_component
│   ├── create_readme.sh
│   ├── create_resources
│   ├── project
│   ├── run_benchmark
│   └── sync_resources.sh
├── src
│   ├── api
│   ├── control_methods
│   ├── data_processors
│   ├── methods
│   ├── metrics
│   └── workflows
└── _viash.yaml
```

```python
import anndata as ad

# Fetch dataset from somewhere.
adata = ad.read_h5ad(\
  "resources_test/common/cxg_mouse_pancreas_atlas/dataset.h5ad")

# Process dataset: Divide the dataset into train and test sets.
test_batch = adata.obs["batch"].to_list()[0]
train_adata = adata[adata.obs["batch"] != test_batch].copy()
test_adata = adata[adata.obs["batch"] == test_batch].copy()

# Method: Apply logistic regression
import sklearn.linear_model
classifier = sklearn.linear_model.LogisticRegression()
classifier.fit(train_adata.obsm["X_pca"],
train_adata.obs["cell_type"].astype(str))
obs_label_pred = classifier.predict(test_adata.obsm["X_pca"])

# Metric: Compute accuracy
import numpy as np
accuracy = np.mean(test_adata.obs["cell_type"] == obs_label_pred)
accuracy # 0.9478672985781991
```

https://openproblems.bio/documentation/create_task/create_components

Step 10: Create components

```yaml
# Component interface
__merge__: /src/api/comp_method.yaml
# Metadata
name: logistic_regression
label: Logistic Regression
summary: Logistic Regression with 100-dimensional PCA coordinates
description: |
  Logistic Regression estimates parameters of a logistic function for [...]
references:
  doi: 123456/123456.1234.789456
links:
  repository: https://github.com/scikit-learn/scikit-learn
  documentation: https://scikit-learn.org/stable/[...]
# Code
resources:
  - type: python_script
    path: script.py
# Reproducibility
engines:
  - type: docker
    image: openproblems/base_python:1.0.0
    setup:
      - type: python
        packages: scikit-learn
# Usability
runners:
  - type: executable
  - type: nextflow
    directives:
      label: [midtime, midmem, lowcpu]
```

```python
import anndata as ad
import sklearn.linear_model
import pandas as pd

print('Reading input files', flush=True)
input_train = ad.read_h5ad(par['input_train'])
input_test = ad.read_h5ad(par['input_test'])

print('Train model', flush=True)
classifier = sklearn.linear_model.LogisticRegression()
classifier.fit(input_train.obsm["X_pca"], input_train.obs["label"].astype(str))

print('Generate predictions', flush=True)
obs_label_pred = classifier.predict(input_test.obsm["X_pca"])

print("Create output data structure", flush=True)
output = ad.AnnData(
  obs=pd.DataFrame(
    { 'label_pred': obs_label_pred },
    index=input_test.obs.index,
  ),
  uns={
    'dataset_id': input_train.uns['dataset_id'],
    'normalization_id': input_train.uns['normalization_id'],
    'method_id': meta['name']
  },
)

print("Write output AnnData to file", flush=True)
output.write_h5ad(par['output'], compression='gzip')
```

https://openproblems.bio/documentation/create_task/create_components

Step 11: Build executables and workflows

```
tree -L 2                ✔
├── CHANGELOG.md
├── common                    # git submodule with common resources
├── CONTRIBUTING.md
├── LICENSE
├── main.nf
├── nextflow.config
├── README.md
├── resources_test            # test resources
│   ├── common                # common test resources you get after running sync_resources
│   └── task_template         # task-specific test resources you need to generate
├── scripts
│   ├── create_component      # generator script for creating a new method or metric
│   ├── create_readme.sh      # generate README from AI files
│   ├── create_resources      # scripts to generate test resources
│   ├── project               # build and test scripts
│   ├── run_benchmark         # run the benchmark
│   └── sync_resources.sh     # fetch test resources
├── src
│   ├── api                   # the api yamls
│   ├── control_methods       # control method components
│   ├── data_processors       # dataset processor
│   ├── methods               # method components
│   ├── metrics               # metrics components
│   └── workflows             # viash nextflow workflow
├── target
│   ├── executable            # standalone executables <-------------------------------------------------------
│   └── nextflow              # nextflow modules & modules <---------------------------------------------------
└── _viash.yaml               # project config
```

https://openproblems.bio/documentation/create_task/create_components

Step 11: Build executables and workflows

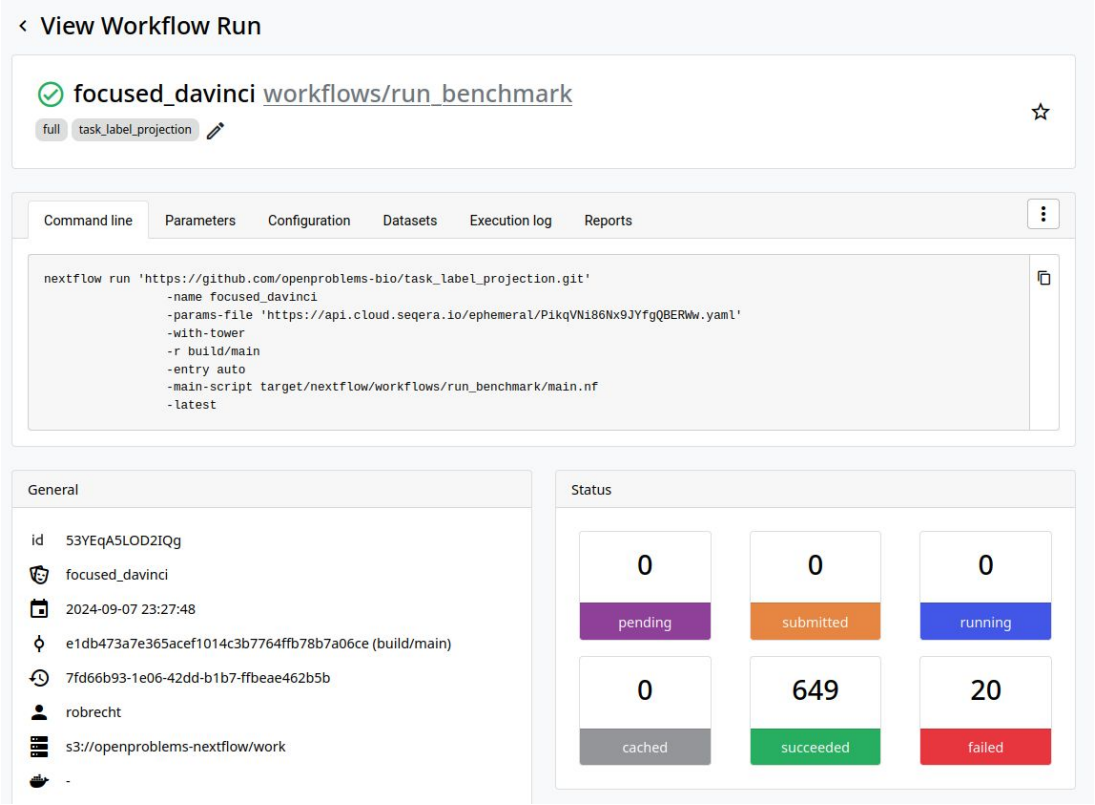
```
scripts/project/build_all_components.sh
# Exporting true_labels (control_methods)       =nextflow=> target/nextflow/control_methods/true_labels
# Exporting process_datasets (workflows)        =nextflow=> target/nextflow/workflows/process_datasets
# Exporting run_benchmark (workflows)           =nextflow=> target/nextflow/workflows/run_benchmark
# Exporting logistic_regression (methods)       =nextflow=> target/nextflow/methods/logistic_regression
# Exporting accuracy (metrics)                  =nextflow=> target/nextflow/metrics/accuracy
# Exporting process_dataset (data_processors)   =nextflow=> target/nextflow/data_processors/process_dataset
# All 10 configs built successfully

tree target
# target
# ├── executable
# │   ├── control_methods/true_labels/true_labels
# │   ├── data_processors/process_dataset
# │   │   ├── process_dataset
# │   │   └── subset_h5ad_by_format.py
# │   ├── methods/logistic_regression/logistic_regression
# │   └── metrics/accuracy/accuracy
# └── nextflow
#     ├── control_methods/true_labels
#     │   ├── main.nf
#     │   └── nextflow.config
#     ├── data_processors/process_dataset/[...]
#     ├── methods/logistic_regression/[...]
#     ├── metrics/accuracy/[...]
#     └── workflows
#         ├── process_datasets/[...]
#         └── run_benchmark/[...]
```

https://openproblems.bio/documentation/create_task/create_workflow

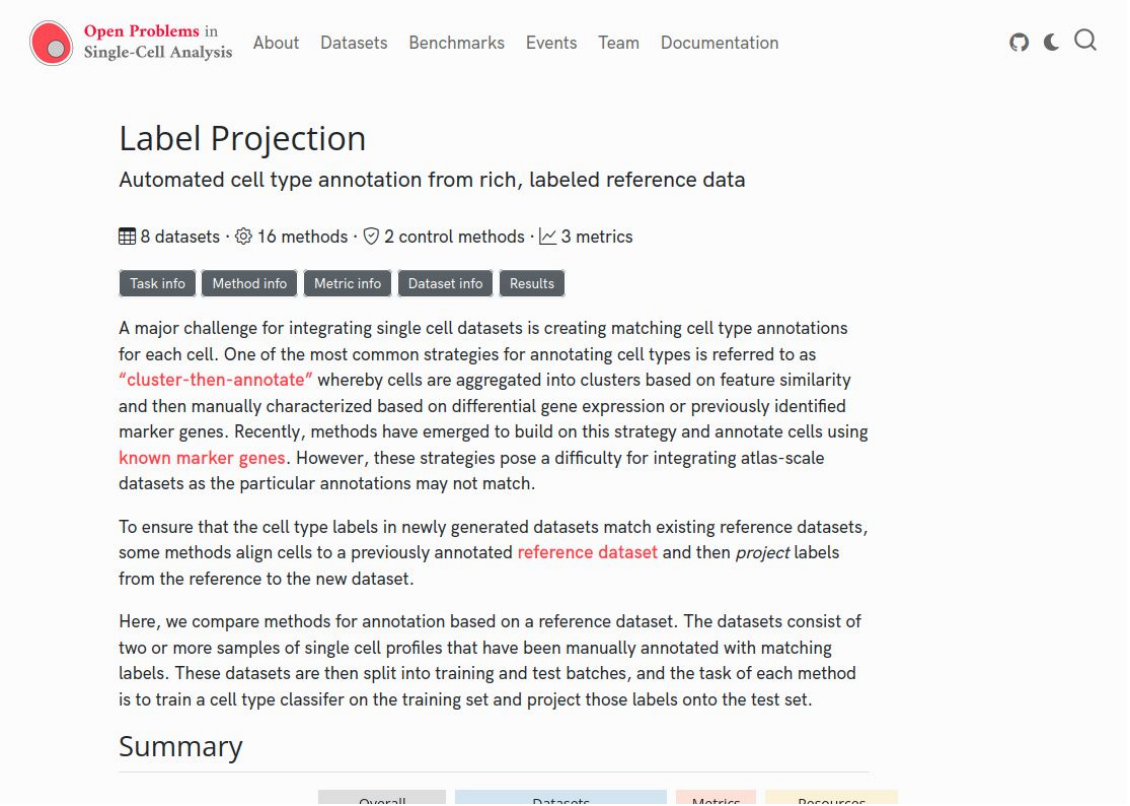
Step 12: Adapt workflow and run it

```
scripts/run_benchmark/run_test_local.sh

# Launching `target/nextflow/workflows/run_benchmark/main.nf` [lonely_hilbert] DSL2 - revision: 5e77baf437
#
# executor >  local (1)
# [56/a5bc85] aut…ract_uns_metadata:processWf:extract_uns_metadata_process (pancreas) [100%] 1 of 1  ✔
# [a9/503cfc] aut…Wf:true_labels:processWf:true_labels_process (pancreas.true_labels) [100%] 1 of 1  ✔
# [29/8a3b51] aut…rocessWf:logistic_regression_process (pancreas.logistic_regression) [100%] 1 of 1  ✔
# [6f/4adf41] aut…accuracy:processWf:accuracy_process (pancreas.true_labels.accuracy) [100%] 2 of 2  ✔
# [b5/1590ec] aut…es:processWf:extract_scores_process (pancreas.true_labels.accuracy) [100%] 2 of 2  ✔
# [0a/8be232] auto:run_benchmark:publishStatesSimpleWf:publishStatesProc (output)     [100%] 1 of 1  ✔

tree resources/results/testrun_2024-09-11_13-41-59
# resources/results/testrun_2024-09-11_13-41-59
# ├── dataset_uns.yaml
# ├── method_configs.yaml
# ├── metric_configs.yaml
# ├── score_uns.yaml
# ├── state.yaml
# └── task_info.yaml
#
# 1 directory, 6 files
```

After a local run on test data succeeds, we run in on our AWS batch



After results are generated,
a results page on the website



After results are generated,
a results page on the website

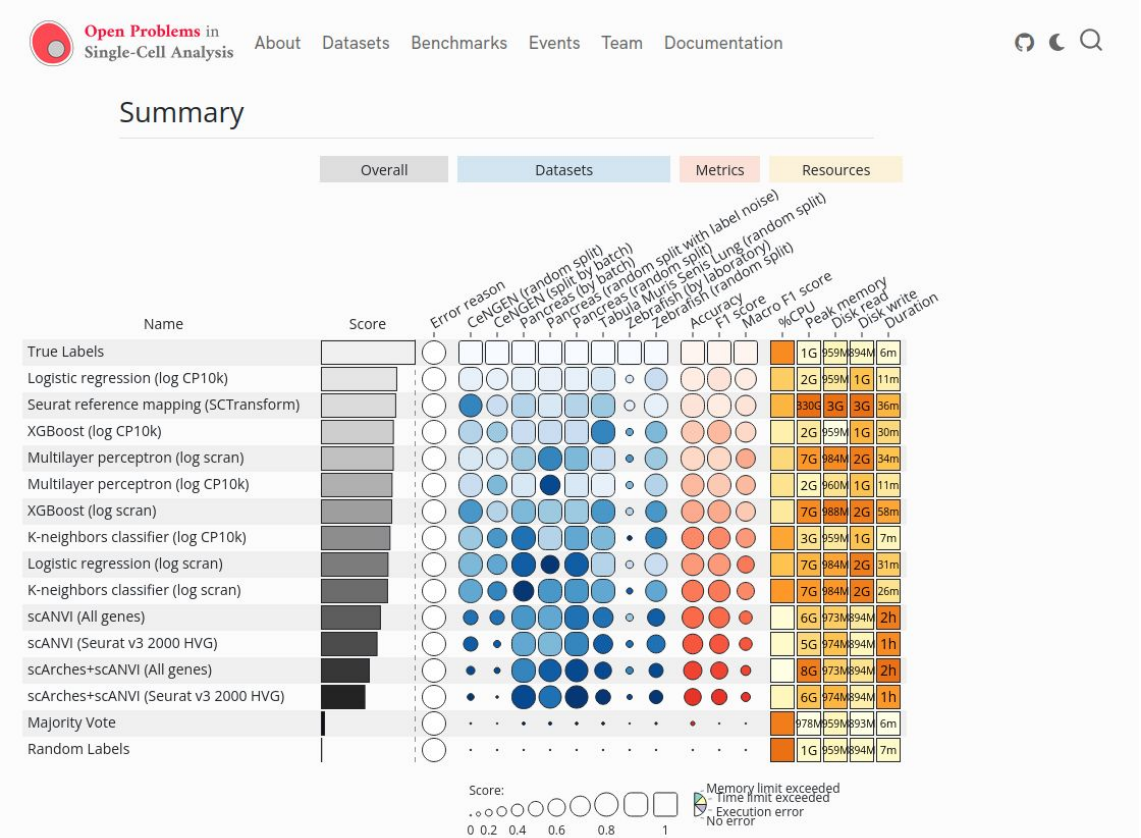
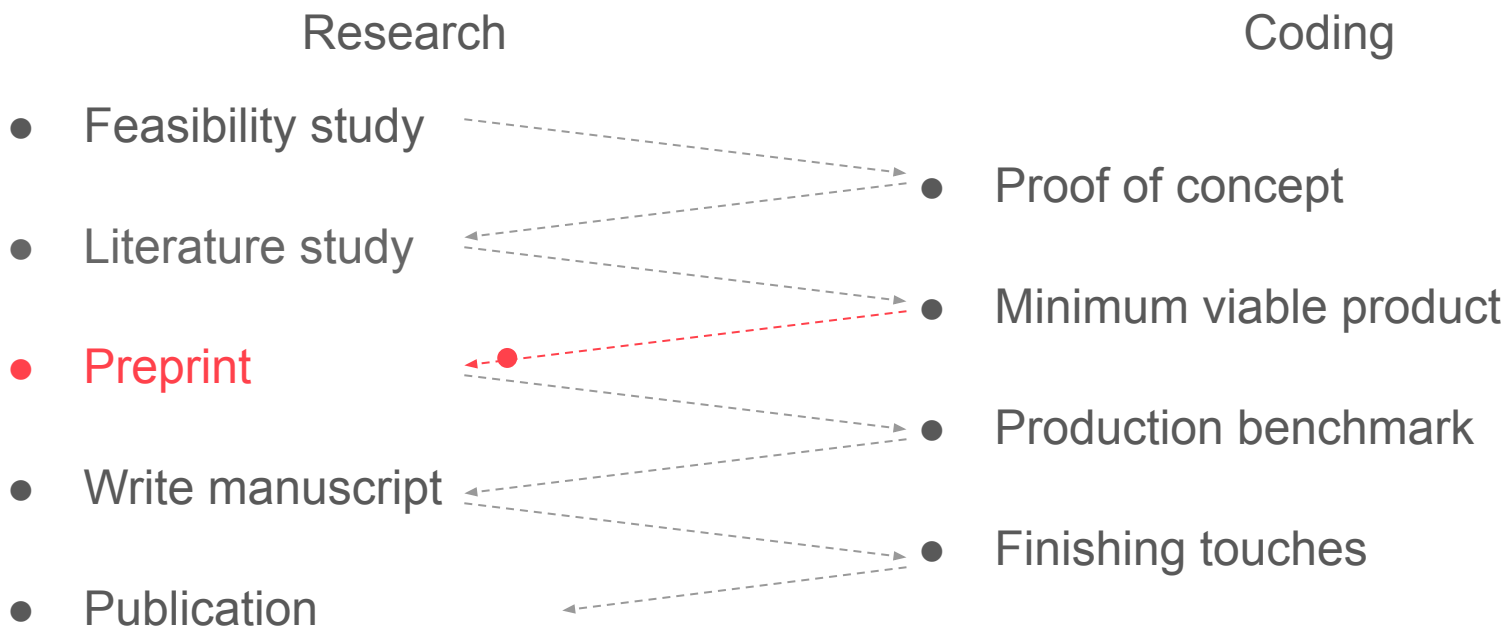
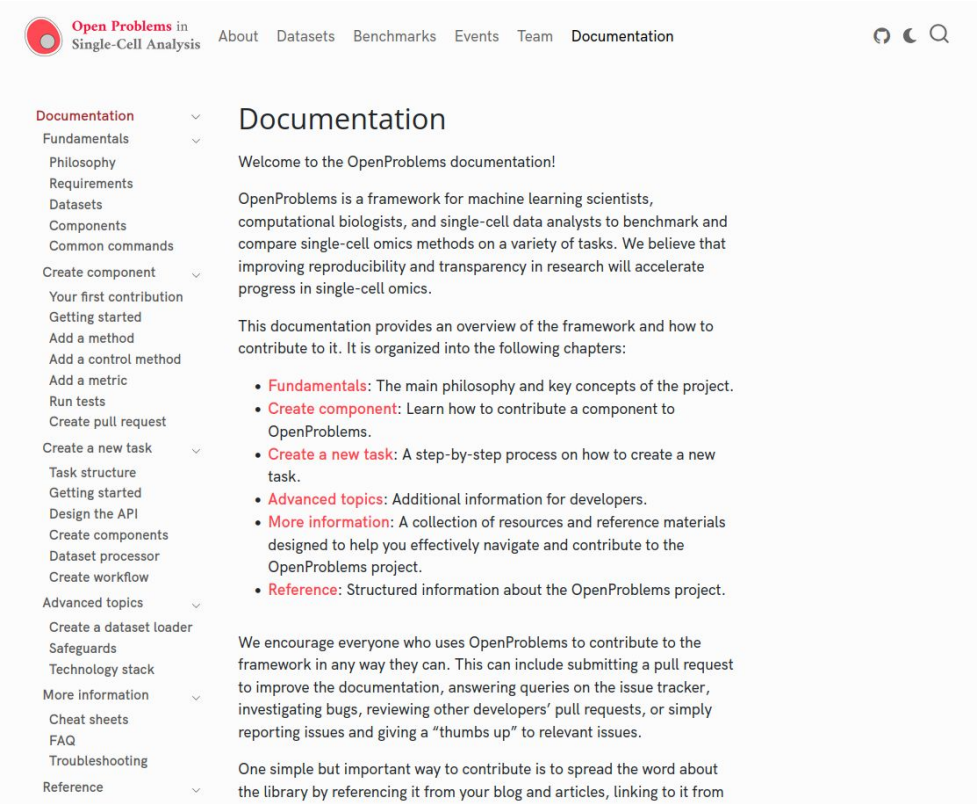


Figure 1: Overview of the results per method. This figures shows the mean of the scaled scores (group Overall), the mean scores per dataset (group Dataset) and the mean scores per metric (group Metric).


Research
Coding
Feasibility study
Literature study
Preprint
Write manuscript
Publication
Proof of concept
Minimum viable product
Production benchmark
Finishing touches

This ended up not being an interactive workshop...

- Documentation at openproblems.bio/documentation
- Recordings of the full workshop planned
- Community involvement:
  - Discord
  - Weekly Friday working meetings
  - openproblems.bio/events/weekly_coworking



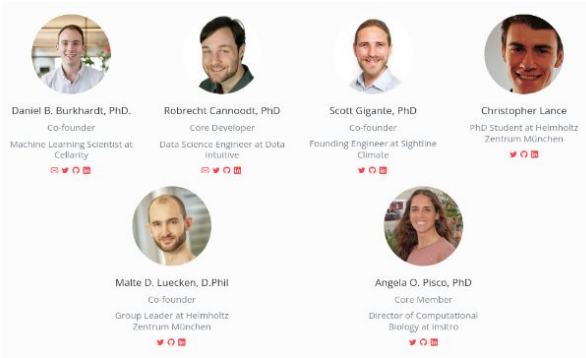
Roadmap

- Fellowship program
- Jamborees coming soon
- Changes to organisation structure:
  - Restructure common datasets per modality
  - Migrating benchmarks
    from openproblems-bio/openproblems
    to openproblems-bio/task_*
- Overhaul website:
  - New design
  - Backend DB
  - Versioned task results

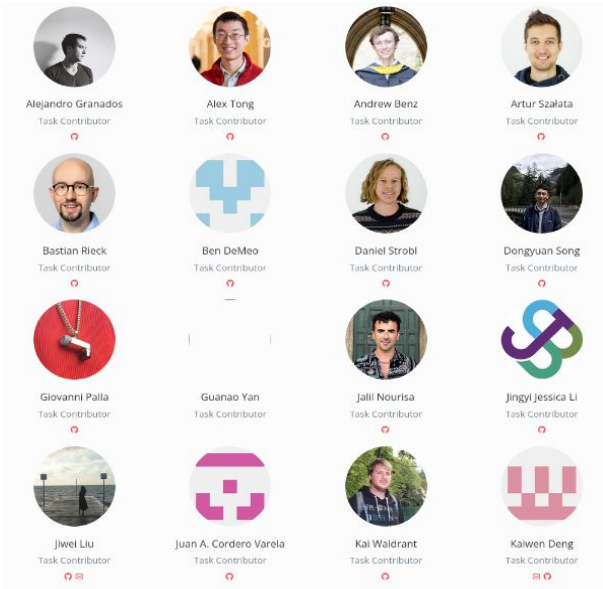
Join our Discord

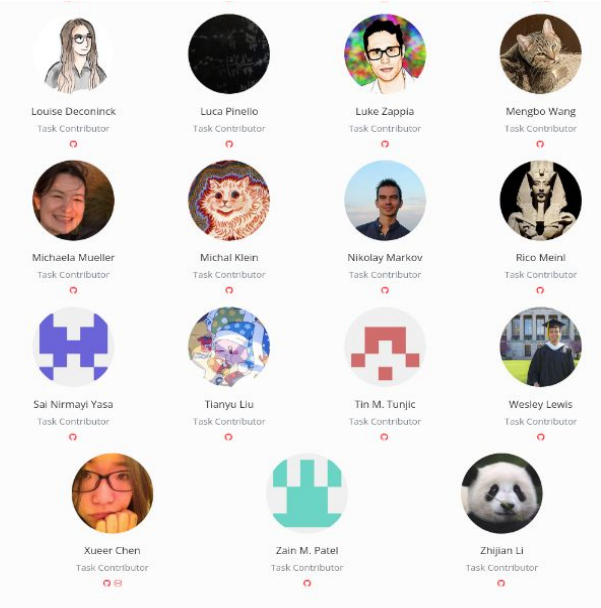
Acknowledgements

Core team



Task contributors[1]





Scientific advisors

Sponsors

Chan Zuckerberg Initiative

Cellarity
A Flagship Pioneering Company

HELMHOLTZ MUNICH

CHAN ZUCKERBERG Biohub San Francisco

immunai

DATA INTUITIVE

[1]: Reach out if your name is missing!