IGVF CRISPR Jamboree 2024: Perturb-seq Inference (R)

Gene Katsevich

February 9, 2024

1 Overview

1.1 Perturb-seq inference

The goal of perturb-seq inference is to quantify the extent to which the perturbation of a given genomic element impacts the expression of a given gene. We allow a range of statistical interpretations of this task. In a frequentist framework, this task can be viewed as testing the null hypothesis that the perturbation of the genomic element has no effect on the gene's expression, or as estimating the effect size of the perturbation on the gene's expression. In a Bayesian framework, this task can be viewed as estimating the posterior probability of the presence of a non-zero effect, or as a posterior mean of the effect size.

1.2 Jamboree goals

The goal of the perturb-seq inference portion of the Jamboree is to implement a number of perturb-seq inference methodologies using common input and output formats. Following the Jamboree, these implementations will be added as modules to a Nextflow pipeline, which will then be used to benchmark their statistical and computational performance. This benchmarking effort will suggest best practices for perturb-seq inference, and will be used to inform the development of the IGVF perturb-seq analysis pipeline.

1.3 Data format overview

The primary input to a perturb-seq inference module is a `MuData` object, which contains both the perturb-seq data and a set of element-gene pairs for which the inference is to be performed. The output of each method should be the same `MuData` object, except with an additional table containing one or more measures of association for each element-gene pair. The `MuData` format is an HDF5-based language-agnostic format compatible with import into both R and Python. Each `MuData` object will contain a minimal set of fields required for inference, and potentially one or more optional fields that provide additional information. For the purposes of this Jamboree, we have provided `MuData` objects for subsets of the Gasperini et al (2019) and Papalexi et al (2021) datasets. For each dataset, we have provided a minimal `MuData` object that contains just the required fields, as well as a more fleshed out object that contains additional optional fields.

1.4 Requested function API, documentation, and demonstration

Please write a function in your language of choice with the following arguments:

- The first argument should be `mudata_input_fp`, a filepath to a `MuData` object.
- The second argument should be `mudata_output_fp`, a filepath to an output `MuData` object.
- There may be one or more additional arguments specific to your method.

The function should read the `MuData` object from `mudata_input_fp`, perform the inference, and write the resulting `MuData` object to `mudata_output_fp` (in R, via `MuData::readH5MU()`). The function should include documentation of any additional arguments used. Furthermore, please include a demonstration of the use of your function on at least one of the sample datasets provided, and a brief discussion of the results.

2 MuData format

Let us walk through the input and output format specifications, from the perspective of R, using a subset of the Gasperini et al (2019) dataset as an example.

```r
library(MuData)
library(SummarizedExperiment)
```

2.1 Required input fields

We start with an example of the minimal `MuData` object required for perturb-seq inference.

```r
mudata_input_fp <- "data/gasperini_inference_input_minimal.h5mu"
input_minimal <- readH5MU(mudata_input_fp)
input_minimal
```

```
## A MultiAssayExperiment object of 2 listed
##  experiments with user-defined names and respective classes.
##  Containing an ExperimentList class object of length 2:
##  [1] gene: SummarizedExperiment with 112 rows and 9704 columns
##  [2] guide: SummarizedExperiment with 55 rows and 9704 columns
## Functionality:
##  experiments() - obtain the ExperimentList instance
##  colData() - the primary/phenotype DataFrame
##  sampleMap() - the sample coordination DataFrame
##  `$`, `[`, `[[` - extract colData columns, subset, or experiment
##  *Format() - convert into a long or wide DataFrame
##  assays() - convert ExperimentList to a SimpleList of matrices
##  exportClass() - save data to flat files
```

The minimal `MuData` object for perturb-seq inference contains two modalities: `gene` and `guide`.

2.1.1 `gene` modality

The minimal `gene` modality needs to just have one assay, whose name is the empty string, containing the RNA UMI counts:

```r
input_minimal[['gene']]
```

```
## class: SummarizedExperiment
## dim: 112 9704
## metadata(0):
## assays(1): ''
## rownames(112): ENSG00000008853 ENSG00000104679 ... ENSG00000198899
##   ENSG00000228253
## rowData names(0):
## colnames(9704): GCTTGAATCGAATGCT-1_1B_1_SI-GA-F2
##   AGCTTGATCGAGAGCA-1_1A_2_SI-GA-E3 ... GGATTACCATGTTGAC-1_2A_4_SI-GA-G5
##   GTGCTTCTCGGATGTT-1_2A_1_SI-GA-G2
## colData names(0):
```

To bring it more in line with the typical `SummarizedExperiment`, we can rename the assay as `counts`:

```r
assayNames(input_minimal[['gene']]) <- 'counts'
input_minimal[['gene']]
```

```
## class: SummarizedExperiment
## dim: 112 9704
## metadata(0):
## assays(1): counts
## rownames(112): ENSG00000008853 ENSG00000104679 ... ENSG00000198899
##   ENSG00000228253
## rowData names(0):
## colnames(9704): GCTTGAATCGAATGCT-1_1B_1_SI-GA-F2
##   AGCTTGATCGAGAGCA-1_1A_2_SI-GA-E3 ... GGATTACCATGTTGAC-1_2A_4_SI-GA-G5
##   GTGCTTCTCGGATGTT-1_2A_1_SI-GA-G2
## colData names(0):
```

```
assay(input_minimal[['gene']], 'counts')[1:2,1:2]
```

```
## 2 x 2 sparse Matrix of class "dgCMatrix"
##                 GCTTGAATCGAATGCT-1_1B_1_SI-GA-F2
## ENSG00000008853                                .
## ENSG00000104679                                2
##                 AGCTTGATCGAGAGCA-1_1A_2_SI-GA-E3
## ENSG00000008853                                .
## ENSG00000104679                                .
```

2.1.2 guide modality

After renaming the first assay of the `guide` modality, this modality must have assays `counts` and `guide_assignment` containing the gRNA UMI counts and binary gRNA assignments, respectively:

```
assayNames(input_minimal[['guide']])[[1]] <- 'counts'
input_minimal[['guide']]
```

```
## class: SummarizedExperiment
## dim: 55 9704
## metadata(2): capture_method moi
## assays(2): counts guide_assignment
## rownames(55): ATGTAGAAGGAGACACCGGG GCGCAGAGGCGGATGTAGAG ...
##   AATCCTCTAATGGACGAAGA ATATTCAGCAGCTAAAGCAT
## rowData names(2): targeting intended_target_name
## colnames(9704): GCTTGAATCGAATGCT-1_1B_1_SI-GA-F2
##   AGCTTGATCGAGAGCA-1_1A_2_SI-GA-E3 ... GGATTACCATGTTGAC-1_2A_4_SI-GA-G5
##   GTGCTTCTCGGATGTT-1_2A_1_SI-GA-G2
## colData names(0):
```

We can view a couple rows and columns of each:

```
cell_ids <- c("GCTTGAATCGAATGCT-1_1B_1_SI-GA-F2",
             "GGTGAAGCACCAGGCT-1_1A_6_SI-GA-E7")
grna_ids <- c("GCCCTGCTACCCACTTACAG",
              "ATGTAGAAGGAGACACCGGG")

assay(input_minimal[['guide']], 'counts')[grna_ids, cell_ids]
```

```
## 2 x 2 sparse Matrix of class "dgCMatrix"
##                      GCTTGAATCGAATGCT-1_1B_1_SI-GA-F2
## GCCCTGCTACCCACTTACAG                                9
## ATGTAGAAGGAGACACCGGG                                .
##                      GGTGAAGCACCAGGCT-1_1A_6_SI-GA-E7
## GCCCTGCTACCCACTTACAG                                .
## ATGTAGAAGGAGACACCGGG                               18
```

```
assay(input_minimal[['guide']], 'guide_assignment')[grna_ids, cell_ids]
```

```
## 2 x 2 sparse Matrix of class "dgCMatrix"
##                      GCTTGAATCGAATGCT-1_1B_1_SI-GA-F2
## GCCCTGCTACCCACTTACAG                                1
## ATGTAGAAGGAGACACCGGG                                .
##                      GGTGAAGCACCAGGCT-1_1A_6_SI-GA-E7
## GCCCTGCTACCCACTTACAG                                .
## ATGTAGAAGGAGACACCGGG                                1
```

In addition to the guide UMI counts and assignments, the `guide` modality must contain certain metadata information. This includes a `rowData()` data frame containing at least the binary variable `targeting` (`TRUE` if the guide targets a genomic element of interest or `FALSE` if it is safe- or non-targeting) and the string `intended_target_name` (the name of the genomic element targeted by the guide).

```
rowData(input_minimal[['guide']])[c(1, 2, 21, 22, 31, 32),]
```

```
## DataFrame with 6 rows and 2 columns
##                        targeting intended_target_name
##                      <character>          <character>
## ATGTAGAAGGAGACACCGGG        TRUE      ENSG00000012660
## GCGCAGAGGCGGATGTAGAG        TRUE      ENSG00000012660
## ACACCCTCATTAGAACCCAG        TRUE      candidate_enh_1
## TTAAGAGCCTCGGTTCCCCT        TRUE      candidate_enh_1
## GACCTCCTGTGATCAGGTGG       FALSE        non-targeting
## ATTGGTATCCGTATAAGCAG       FALSE        non-targeting
```

Note that the `targeting` column is a string rather than a Boolean due to type compatibility issues involving R, Python, and HDF5. It can be cast to a Boolean if desired.

Finally, the guide modality must contain `metadata()` fields called `moi` (`low` or `high`) and `capture_method` (`"CROP-seq"` or `"direct capture"`):

```
metadata(input_minimal[['guide']])
```

```
## $capture_method
## [1] "CROP-seq"
##
## $moi
## [1] "high"
```

2.1.3 Global `metadata`

The input `MuData` object is also required to have a global `metadata` field named `pairs_to_test`, which is a data frame containing the pairs of elements (specified via `intended_target_name`) and genes (specified via `gene_id`) for which the inference is to be performed.

```
metadata(input_minimal)$pairs_to_test |> as.data.frame() |> head()
```

```
##           gene_id intended_target_name
## 1 ENSG00000187109      ENSG00000187109
## 2 ENSG00000114850      ENSG00000114850
## 3 ENSG00000134851      ENSG00000134851
## 4 ENSG00000163866      ENSG00000163866
## 5 ENSG00000181610      ENSG00000181610
## 6 ENSG00000113552      ENSG00000113552
```

2.2 Optional input fields

Next we consider optional fields that can be included in the input MuData object.

```
mudata_input_fp = "data/gasperini_inference_input.h5mu"
input_optional = readH5MU(mudata_input_fp)
input_optional
```

```
## A MultiAssayExperiment object of 2 listed
##  experiments with user-defined names and respective classes.
##  Containing an ExperimentList class object of length 2:
##  [1] gene: SummarizedExperiment with 112 rows and 9704 columns
##  [2] guide: SummarizedExperiment with 55 rows and 9704 columns
## Functionality:
##  experiments() - obtain the ExperimentList instance
##  colData() - the primary/phenotype DataFrame
##  sampleMap() - the sample coordination DataFrame
##  `$`, `[`, `[[` - extract colData columns, subset, or experiment
##  *Format() - convert into a long or wide DataFrame
##  assays() - convert ExperimentList to a SimpleList of matrices
##  exportClass() - save data to flat files
```

2.2.1 gene modality

The MuData object may include cellwise covariates for the gene modality in colData(), such as number of genes with nonzero UMI counts (num_expressed_genes) and total RNA UMIs (total_gene_umis):

```
colData(input_optional[['gene']])
```

```
## DataFrame with 9704 rows and 2 columns
##                                  num_expressed_genes total_gene_umis
##                                            <integer>       <numeric>
## GCTTGAATCGAATGCT-1_1B_1_SI-GA-F2                  41             280
## AGCTTGATCGAGAGCA-1_1A_2_SI-GA-E3                  35             192
## CCCAATCTCCTCAATT-1_1B_1_SI-GA-F2                  41             781
## CGCGGTACACTGTCGG-1_1A_2_SI-GA-E3                  37             189
## GGACGTCTCATGTCTT-1_1B_8_SI-GA-F9                  32             262
## ...                                              ...             ...
## CGCTATCTCTATCGCC-1_2A_4_SI-GA-G5                  23             203
## TCACAAGCAGCCTTGG-1_2A_6_SI-GA-G7                  30             173
## GCTGCAGGTGAAGGCT-1_2B_6_SI-GA-H7                  37             428
## GGATTACCATGTTGAC-1_2A_4_SI-GA-G5                  47             658
## GTGCTTCTCGGATGTT-1_2A_1_SI-GA-G2                  23             166
```

The MuData object may include per-gene metadata in rowData(), such as the HGNC gene symbol (symbol), the gene chromosome (chr), start (gene_start), and end (gene_end) coordinates:

```
rowData(input_optional[['gene']])
```

```
## DataFrame with 112 rows and 4 columns
##                      symbol    gene_chr gene_start  gene_end
##                 <character> <character>  <numeric> <numeric>
## ENSG00000008853     RHOBTB2        chr8   22844930  22844931
## ENSG00000104679      R3HCC1        chr8   23145421  23145422
## ENSG00000104689   TNFRSF10A        chr8   23082573  23082574
## ENSG00000120889   TNFRSF10B        chr8   22926533  22926534
## ENSG00000120896      SORBS3        chr8   22409208  22409209
## ...                          ...          ...        ...        ...
## ENSG00000114850             SSR3         chr3  156271913 156271914
## ENSG00000072274             TFRC         chr3  195808960 195808961
## ENSG00000134851          TMEM165         chr4   56262124  56262125
## ENSG00000198899                                       NaN       NaN
## ENSG00000228253                                       NaN       NaN
```

2.2.2 guide modality

The `MuData` object may include cellwise covariates for the `guide` modality in `colData()`, such as number of guides with nonzero UMI counts (`num_expressed_guides`) and total guide UMIs (`total_guide_umis`):

```
colData(input_optional[['guide']])
```

```
## DataFrame with 9704 rows and 2 columns
##                                  num_expressed_guides total_guide_umis
##                                             <integer>        <numeric>
## GCTTGAATCGAATGCT-1_1B_1_SI-GA-F2                    1                9
## AGCTTGATCGAGAGCA-1_1A_2_SI-GA-E3                    1               18
## CCCAATCTCCTCAATT-1_1B_1_SI-GA-F2                    1               24
## CGCGGTACACTGTCGG-1_1A_2_SI-GA-E3                    1               26
## GGACGTCTCATGTCTT-1_1B_8_SI-GA-F9                    1               12
## ...                                               ...              ...
## CGCTATCTCTATCGCC-1_2A_4_SI-GA-G5                    1                5
## TCACAAGCAGCCTTGG-1_2A_6_SI-GA-G7                    1               39
## GCTGCAGGTGAAGGCT-1_2B_6_SI-GA-H7                    1               21
## GGATTACCATGTTGAC-1_2A_4_SI-GA-G5                    1               73
## GTGCTTCTCGGATGTT-1_2A_1_SI-GA-G2                    1               12
```

The `MuData` object may include per-guide metadata in `rowData()` in addition to the required `targeting` and `intended_target_name` fields, such as the chromosome (`intended_target_chr`), start (`intended_target_start`), and end (`intended_target_end`) of the targeted element:

```
rowData(input_optional[['guide']])[c(1, 2, 21, 22, 31, 32),]
```

```
## DataFrame with 6 rows and 5 columns
##                        targeting intended_target_name intended_target_chr
##                      <character>          <character>         <character>
## ATGTAGAAGGAGACACCGGG        TRUE      ENSG00000012660                chr6
## GCGCAGAGGCGGATGTAGAG        TRUE      ENSG00000012660                chr6
## ACACCCTCATTAGAACCCAG        TRUE      candidate_enh_1                chr8
## TTAAGAGCCTCGGTTCCCCT        TRUE      candidate_enh_1                chr8
## GACCTCCTGTGATCAGGTGG       FALSE        non-targeting
## ATTGGTATCCGTATAAGCAG       FALSE        non-targeting
##                      intended_target_start intended_target_end
##                                  <numeric>           <numeric>
## ATGTAGAAGGAGACACCGGG              53213723            53213738
## GCGCAGAGGCGGATGTAGAG              53213738            53213754
## ACACCCTCATTAGAACCCAG              23366136            23366564
## TTAAGAGCCTCGGTTCCCCT              23366564            23366992
## GACCTCCTGTGATCAGGTGG                    -9                  -9
## ATTGGTATCCGTATAAGCAG                    -9                  -9
```

2.2.3 Global metadata

Optionally, the `MuData` input object can contain a global `colData()` containing cell-level information that is not specific to modality, such as batch information. Here is what it looks like for the Gasperini data:

```r
colData(input_optional)
```

```
## DataFrame with 9704 rows and 3 columns
##                                  prep_batch   within_batch_chip
##                                 <character>         <character>
## GCTTGAATCGAATGCT-1_1B_1_SI-GA-F2 prep_batch_1 within_batch_chip_B
## AGCTTGATCGAGAGCA-1_1A_2_SI-GA-E3 prep_batch_1 within_batch_chip_A
## CCCAATCTCCTCAATT-1_1B_1_SI-GA-F2 prep_batch_1 within_batch_chip_B
## CGCGGTACACTGTCGG-1_1A_2_SI-GA-E3 prep_batch_1 within_batch_chip_A
## GGACGTCTCATGTCTT-1_1B_8_SI-GA-F9 prep_batch_1 within_batch_chip_B
## ...                                       ...                 ...
## CGCTATCTCTATCGCC-1_2A_4_SI-GA-G5 prep_batch_2 within_batch_chip_A
## TCACAAGCAGCCTTGG-1_2A_6_SI-GA-G7 prep_batch_2 within_batch_chip_A
## GCTGCAGGTGAAGGCT-1_2B_6_SI-GA-H7 prep_batch_2 within_batch_chip_B
## GGATTACCATGTTGAC-1_2A_4_SI-GA-G5 prep_batch_2 within_batch_chip_A
## GTGCTTCTCGGATGTT-1_2A_1_SI-GA-G2 prep_batch_2 within_batch_chip_A
##                                  within_chip_lane
##                                       <character>
## GCTTGAATCGAATGCT-1_1B_1_SI-GA-F2 within_chip_lane_1
## AGCTTGATCGAGAGCA-1_1A_2_SI-GA-E3 within_chip_lane_2
## CCCAATCTCCTCAATT-1_1B_1_SI-GA-F2 within_chip_lane_1
## CGCGGTACACTGTCGG-1_1A_2_SI-GA-E3 within_chip_lane_2
## GGACGTCTCATGTCTT-1_1B_8_SI-GA-F9 within_chip_lane_8
## ...                                             ...
## CGCTATCTCTATCGCC-1_2A_4_SI-GA-G5 within_chip_lane_4
## TCACAAGCAGCCTTGG-1_2A_6_SI-GA-G7 within_chip_lane_6
## GCTGCAGGTGAAGGCT-1_2B_6_SI-GA-H7 within_chip_lane_6
## GGATTACCATGTTGAC-1_2A_4_SI-GA-G5 within_chip_lane_4
## GTGCTTCTCGGATGTT-1_2A_1_SI-GA-G2 within_chip_lane_1
```

2.2.4 Pairs to test

Optionally, the `pairs_to_test` field of the global metadata can have a third column: `pair_type`:

```r
metadata(input_optional)$pairs_to_test |> as.data.frame() |> head()
```

```
##           gene_id intended_target_name        pair_type
## 1 ENSG00000187109      ENSG00000187109 positive_control
## 2 ENSG00000114850      ENSG00000114850 positive_control
## 3 ENSG00000134851      ENSG00000134851 positive_control
## 4 ENSG00000163866      ENSG00000163866 positive_control
## 5 ENSG00000181610      ENSG00000181610 positive_control
## 6 ENSG00000113552      ENSG00000113552 positive_control
```

This optional column classifies pairs based on whether they are intended to be positive controls (an association is known to exist), negative controls (an association is known not to exist), or discovery pairs (pairs where it is unknown whether an association exists). This information need not be used by the inference module, but it is useful for downstream analysis.

2.3 Output fields

The output should be the same `MuData` object as the input, with the addition of a `test_results` field to the global `metadata`:

```r
mudata_output_fp <- "data/gasperini_inference_output.h5mu"
output_optional <- readH5MU(mudata_output_fp)
```

```
output_optional
```

```
## A MultiAssayExperiment object of 2 listed
##  experiments with user-defined names and respective classes.
##  Containing an ExperimentList class object of length 2:
##  [1] gene: SummarizedExperiment with 112 rows and 9704 columns
##  [2] guide: SummarizedExperiment with 55 rows and 9704 columns
## Functionality:
##  experiments() - obtain the ExperimentList instance
##  colData() - the primary/phenotype DataFrame
##  sampleMap() - the sample coordination DataFrame
##  `$`, `[`, `[[` - extract colData columns, subset, or experiment
##  *Format() - convert into a long or wide DataFrame
##  assays() - convert ExperimentList to a SimpleList of matrices
##  exportClass() - save data to flat files
```

```
metadata(output_optional)$test_results |> as.data.frame() |> head()
```

```
##           gene_id intended_target_name    log2_fc      p_value        pair_type
## 1 ENSG00000187109      ENSG00000187109 -0.7743672 3.217223e-85 positive_control
## 2 ENSG00000114850      ENSG00000114850 -1.8495718 2.414163e-79 positive_control
## 3 ENSG00000134851      ENSG00000134851 -0.8938597 4.309833e-50 positive_control
## 4 ENSG00000163866      ENSG00000163866 -1.2236996 4.704066e-49 positive_control
## 5 ENSG00000181610      ENSG00000181610 -1.3142850 3.766690e-42 positive_control
## 6 ENSG00000113552      ENSG00000113552 -1.6785978 2.554175e-39 positive_control
```

This is a data frame containing the same columns as the `pairs_to_test` data frame, plus at least one column containing a measure of the association for each pair. These columns can be `p_value`, `log2_fc`, `posterior_probability`, or any other measure of association.

3 Sample submission

Here we present a sample Jamboree submission.

3.1 Function

Here is a sample function that computes a *p*-value based on a Wilcoxon test:

```
#' Wilcoxon test
#'
#' @param mudata_input_fp Path to input MuData
#' @param mudata_output_fp Path to output MuData
#' @param side The sidedness of the test (`left`, `right`, or `both`)
#'
#' @return This function is called for its side effect of writing the output MuData.
compute_wilcoxon_test <- function(mudata_input_fp, mudata_output_fp, side) {
  # Read input MuData
  mudata <- MuData::readH5MU(mudata_input_fp)
  # Rename primary assay to 'counts'
  if(is.null(SummarizedExperiment::assayNames(mudata[['gene']]))){
    SummarizedExperiment::assayNames(mudata[['gene']]) <- 'counts'
  } else{
    SummarizedExperiment::assayNames(mudata[['gene']])[[1]] <- 'counts'
  }
  SummarizedExperiment::assayNames(mudata[['guide']])[[1]] <- 'counts'
```

```r
# Extract pairs to test and MOI
pairs_to_test <- MultiAssayExperiment::metadata(input_minimal)$pairs_to_test |>
  as.data.frame()
moi <- MultiAssayExperiment::metadata(input_minimal[["guide"]])$moi
# In low-MOI case, extract control cells as those containing an NT gRNA
if (moi == "low") {
  non_targeting_guides <- SummarizedExperiment::rowData(mudata[["guide"]]) |>
    as.data.frame() |>
    dplyr::filter(targeting == "FALSE") |>
    rownames()
  nt_grna_presence <- SummarizedExperiment::assay(
    mudata[["guide"]],
    "guide_assignment"
  )[non_targeting_guides, ] |>
    apply(MARGIN = 2, FUN = max)
  control_cells <- names(nt_grna_presence)[nt_grna_presence == 1]
}
# Initialize test results data frame based on pairs_to_test
test_results <- pairs_to_test |>
  dplyr::mutate(p_value = NA_real_)
# Carry out the Wilcoxon test for each pair
for (pair_idx in 1:nrow(pairs_to_test)) {
  # Extract the gene and element to be tested
  gene_id <- pairs_to_test[pair_idx, "gene_id"]
  intended_target_name <- pairs_to_test[pair_idx, "intended_target_name"]
  # Find the set of treatment cells (i.e. cells with element targeted)
  grnas_targeting_element <- SummarizedExperiment::rowData(mudata[["guide"]]) |>
    as.data.frame() |>
    dplyr::filter(intended_target_name == !!intended_target_name) |>
    rownames()
  element_targeted <- assay(
    mudata[["guide"]],
    "guide_assignment"
  )[grnas_targeting_element, ] |>
    apply(MARGIN = 2, FUN = max)
  treatment_cells <- names(element_targeted)[element_targeted == 1]
  # Set controls cells using the complement set in high MOI
  if (moi == "high") {
    control_cells <- names(element_targeted)[element_targeted == 0]
  }
  # extract expressions for treatment and control cells
  treatment_expressions <- SummarizedExperiment::assay(
    mudata[["gene"]],
    "counts"
  )[gene_id, treatment_cells]
  control_expressions <- SummarizedExperiment::assay(
    mudata[["gene"]],
    "counts"
  )[gene_id, control_cells]
  # Map `side` argument to `alternative` argument required by `wilcox.test()`
  alternative <- switch(side,
    left = "less",
    right = "greater",
      both = "two.sided"
    )
    # Carry out the Wilcoxon test
    test_results[pair_idx, "p_value"] <- stats::wilcox.test(
      x = treatment_expressions,
      y = control_expressions,
      alternative = alternative
    )$p.value
  }
  # Add output to MuData and write to disk
  mudata_output <- mudata
  MultiAssayExperiment::metadata(mudata_output)$test_results <- test_results
  MuData::writeH5MU(mudata_output, mudata_output_fp)
}
```

3.2 Demonstration

Here is a demonstration of this function on the Gasperini data:

```
# Compute Wilcoxon test on the Gasperini data
compute_wilcoxon_test(
  mudata_input_fp = "data/gasperini_inference_input.h5mu",
  mudata_output_fp = "data/gasperini_inference_output_wilcoxon.h5mu",
  side = "left"
)

# Read results from disk
output_wilcoxon <- MuData::readH5MU("data/gasperini_inference_output_wilcoxon.h5mu")

# Print results
metadata(output_wilcoxon)$test_results |> as.data.frame() |> head()
```

```
##           gene_id intended_target_name       p_value
## 1 ENSG00000187109      ENSG00000187109  3.030394e-72
## 2 ENSG00000114850      ENSG00000114850 4.672505e-128
## 3 ENSG00000134851      ENSG00000134851  6.398416e-49
## 4 ENSG00000163866      ENSG00000163866  4.183842e-68
## 5 ENSG00000181610      ENSG00000181610  3.903196e-65
## 6 ENSG00000113552      ENSG00000113552  1.360795e-60
```