Precision Health with Multi-Omics and Environmental Data: A Primer

Jesse A. Goodrich, Hongxu Wang, Qiran Jia, & David Conti

2023-11-02

Table of contents

Preface

Integrating environmental data with biological data from multiple omics datasets can help provide unprecedented insights into the complex interplay of environment and biology. Joint analysis of environment and multi-omics can provide a more comprehensive picture of individual health and disease than individually analyzing datasets. However, to harness the full potential of using multi-omics data to understand environmental and biological drivers of disease, researchers need a robust framework for understanding how and why to perform different multiomic integration techniques.

This book is intended as a resource for researchers aiming to incorporate omics datasets to understand how environmental or biological factors impact human health and disease. We aim to explain the concepts, techniques, and methodologies that allow researchers to fully leverage the information in multidimensional datasets to obtain biologically relevant and actionable insights into environmental and biological impacts on disease.

This book has three sections corresponding to each column shown in Figure 1. In Chapter 2, we provide an example of high dimensional mediation with early, intermediate, and late integration, as shown in the first column of Figure 1. In Chapter 3, we provide an example of mediation with latent factors, a two-step approach that first uses dimensionality reduction on the omics datasets and then performs mediation on the resulting factors, as shown in the second column of Figure 1. In Chapter 4, we provide an example of quasi/intermediate integration, an approach where information on environmental factors and information on multiple omic layers are analyzed jointly in a single unified analysis, as shown in column 3 of Figure 1.

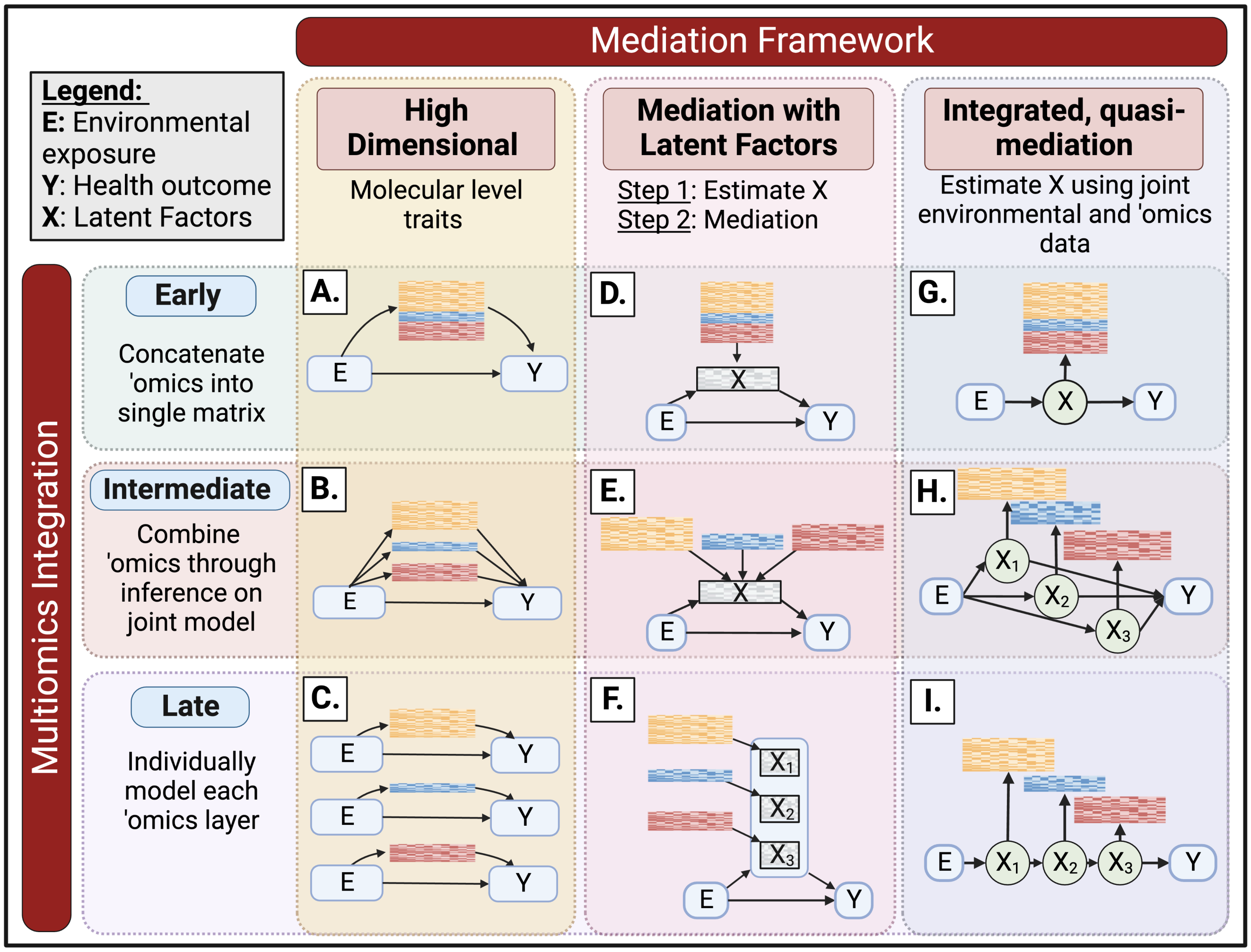

Mediation Framework
Legend:
E: Environmental exposure
Y: Health outcome
X: Latent Factors
High Dimensional
Molecular level traits
Mediation with Latent Factors
Step 1: Estimate X
Step 2: Mediation
Integrated, quasi-mediation
Estimate X using joint environmental and 'omics data
Multiomics Integration
Early
Concatenate 'omics into single matrix
Intermediate
Combine 'omics through inference on joint model
Late
Individually model each 'omics layer
A.
B.
C.
D.
E.
F.
G.
H.
I.
E
Y
X
X1
X2
X3

Figure 1: Conceptual diagram illustrating the analytic framework for mediation analysis with multiple omic layers.

1 Introduction

1.1 R Packages

The following is a list of packages that are used throughout this book that need to be loaded before any analysis. A complete list of all the packages used in the book, as well as code to download these packages, can be found in Chapter 5.

```r
# General Packages:
library(EnvirOmix)
library(tidyverse)
library(table1)
# Load plotting packages:
library(ggplot2)
library(cowplot)
library(ComplexHeatmap)
library(ggh4x)
# Packages for Quasi-mediation:
#LUCIDus Version 3.0.1 is required for this analysis:
library(LUCIDus)
library(networkD3)
library(plotly)
library(htmlwidgets)
library(jsonlite)
```

In order to replicate the style of the figures in this book, you will also have to set the ggplot theme:

```r
ggplot2::theme_set(cowplot::theme_cowplot())
```

1.2 The Data

The data used in this project is based off of simulated data from the Human Early Life Exposome (HELIX) cohort (Vrijheid et al. 2014). The data was simulated for one exposure, five omics layers, and one continuous outcome (after publication, this data will be available on github). The format of this data is a named list with 6 elements. It includes separate numeric matrices for each of the 5 omics layers, as well as the exposure and phenotype data. In all datasets in the list, the rows represent individuals and the columns represent omics features. In this analysis, the exposure and outcome are:

- **Exposure**: *hs_hg_m_resid*, representing maternal mercury levels
- **Outcome**: *ck18_scaled*, representing child liver enzyme levels, a major risk factor for non-alcoholic fatty liver disease (NAFLD).

This data is provided in the EnvirOmics package, which you can access in R using the following code:

```r
# Load R package
library(EnvirOmix)

# Load simulated data
data(simulated_data)

# Define exposure and outcome name
exposure <- simulated_data[["phenotype"]]$hs_hg_m_scaled
outcome  <- simulated_data[["phenotype"]]$ck18_scaled

# Get numeric matrix of covariates
covs <- simulated_data[["phenotype"]][c("e3_sex_None", "hs_child_age_yrs_None")]
covs$e3_sex_None <- ifelse(covs$e3_sex_None == "male", 1, 0)

# create list of omics data
omics_lst <- simulated_data[-which(names(simulated_data) == "phenotype")]

# Create data frame of omics data
omics_df <- omics_lst |>
  purrr::map(~as_tibble(.x, rownames = "name")) |>
  purrr::reduce(left_join, by = "name") |>
  column_to_rownames("name")
```

1.2.1 Descriptive Statistics

Table 1.1 shows the summary statistics for the exposure and phenotype data in this analysis.

```r
table1::table1(~., data = simulated_data[["phenotype"]][,-1])
```

Table 1.1: Descriptive Statistics for the Simulated Variables

	Overall
	(N=420)
hs_child_age_yrs_None	
Mean (SD)	7.22 (1.04)
Median [Min, Max]	7.18 [3.93, 10.9]
hs_hg_m_scaled	
Mean (SD)	-0.0148 (1.02)
Median [Min, Max]	-0.0433 [-2.73, 3.34]
ck18_scaled	
Mean (SD)	-0.0511 (1.03)
Median [Min, Max]	0.00883 [-3.56, 3.09]
e3_sex_None	
female	192 (45.7%)
male	228 (54.3%)
h_fish_preg_Ter	
1	233 (55.5%)
2	88 (21.0%)
3	99 (23.6%)

1.2.2 Mercury exposure and childhood MAFLD risk

```r
lm_res <- lm(ck18_scaled ~ hs_hg_m_scaled +
     e3_sex_None +
     hs_child_age_yrs_None,
   data = simulated_data[["phenotype"]])

summary(lm_res)
```

In the simulated data, each 1 standard deviation increase in maternal mercury was associated with a 0.11 standard deviation increase in CK18 enzymes (Figure 1.1; p=0.02), after adjusting for child age and child sex.

```r
ggplot(data = simulated_data[["phenotype"]],
       aes(x = hs_hg_m_scaled, y = ck18_scaled)) +
  geom_point() +
  stat_smooth(method = "lm",
              formula = y ~ x ,
              geom = "smooth") +
  xlab("Maternal Mercury Exposure (Scaled)") +
  ylab("CK-18 Levels (Scaled)")
```

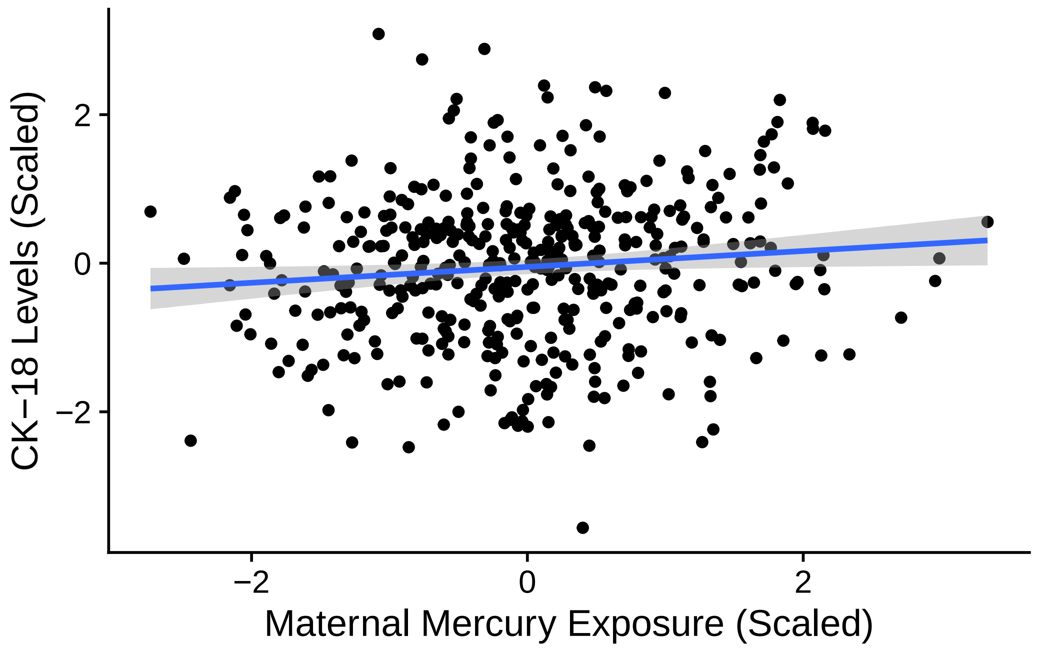


Figure 1.1: Association between maternal mercury and CK18 in the Simulated Data

1.2.3 Correlation of omics features

Figure 1.2 shows the correlation within and between the omics layers in the simulated data.

```r
# Change omics list elements to dataframes
omics_df <- purrr::map(omics_lst, ~as_tibble(.x, rownames = "name")) %>%
  purrr::reduce(left_join, by = "name") %>%
  column_to_rownames("name")

meta_df <- imap_dfr(purrr::map(omics_lst, ~as_tibble(.x)),
                    ~tibble(omic_layer = .y, ftr_name = names(.x)))

# Correlation Matrix
cormat <- cor(omics_df, method = "pearson")

# Annotations
annotation <- data.frame(
  ftr_name = colnames(cormat),
  index = 1:ncol(cormat)) %>%
  left_join(meta_df, by = "ftr_name") %>%
  mutate(omic_layer = str_to_title(omic_layer))

# Make Plot
Heatmap(cormat,
        row_split = annotation$omic_layer,
        column_split = annotation$omic_layer,
        show_row_names = FALSE,
        show_column_names = FALSE,
        column_title_gp = gpar(fontsize = 12),
        row_title_gp = gpar(fontsize = 12),
        heatmap_legend_param = list(title = "Correlation"))
```

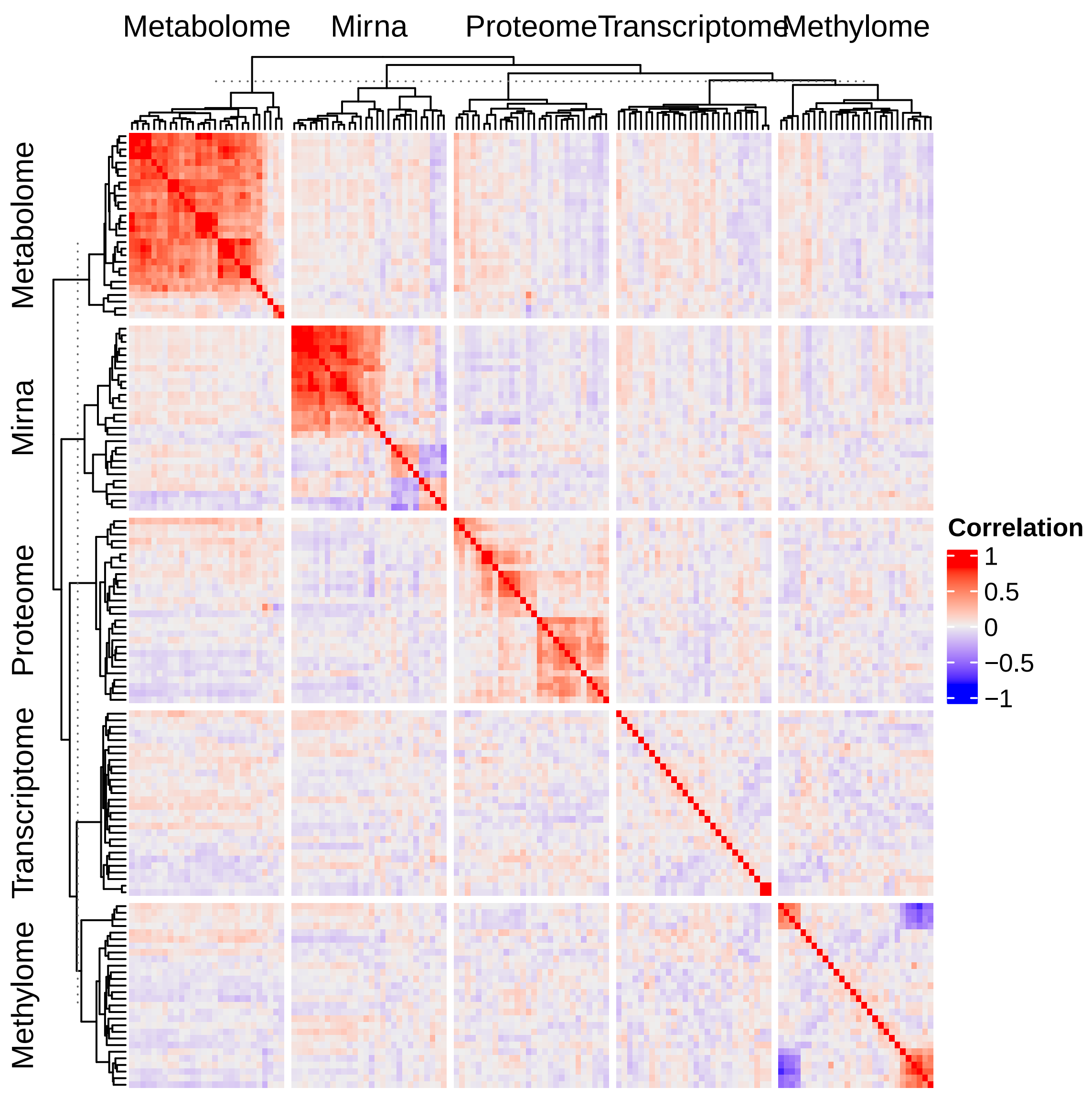


Figure 1.2: Heatmap illustrating the correlation of molecular features within and between different omics layers.

2 High Dimensional Multiomic Mediation

2.1 Load data and packages

Before starting, load the data and packages using the following code.

```r
# Load R package
library(EnvirOmix)

# Load simulated data
data(simulated_data)

# Define exposure and outcome name
exposure <- simulated_data[["phenotype"]]$hs_hg_m_scaled
outcome  <- simulated_data[["phenotype"]]$ck18_scaled

# Get numeric matrix of covariates
covs <- simulated_data[["phenotype"]][c("e3_sex_None", "hs_child_age_yrs_None")]
covs$e3_sex_None <- ifelse(covs$e3_sex_None == "male", 1, 0)

# create list of omics data
omics_lst <- simulated_data[-which(names(simulated_data) == "phenotype")]

# Create data frame of omics data
omics_df <- omics_lst |>
  purrr::map(~as_tibble(.x, rownames = "name")) |>
  purrr::reduce(left_join, by = "name") |>
  column_to_rownames("name")
```

2.2 *Early integration*

High dimensional multiomic mediation with early multiomic integration (Figure 1, panel a) identified differentially methylated CpG sites, gene transcript clusters, metabolites, and proteins which mediated associations of prenatal mercury with MAFLD risk in adolescents (Figure 2.1). Combining all omics layers before analysis identifies the strongest mediating feature across all omics layers without accounting for the differences in underlying correlation structure. For this analysis, we used High Dimensional Mediation Analysis (HIMA), a penalization-based mediation method implemented in the R package HIMA (Zhang et al. 2016).

```r
# Run Analysis
result_hima_early <- hidimum(exposure = exposure,
                             outcome = outcome,
                             omics_lst = omics_lst,
                             covs = covs,
                             Y.family = "gaussian",
                             M.family = "gaussian",
                             integration = "early")
# Plot Result
plot_hidimum(result_hima_early)
```

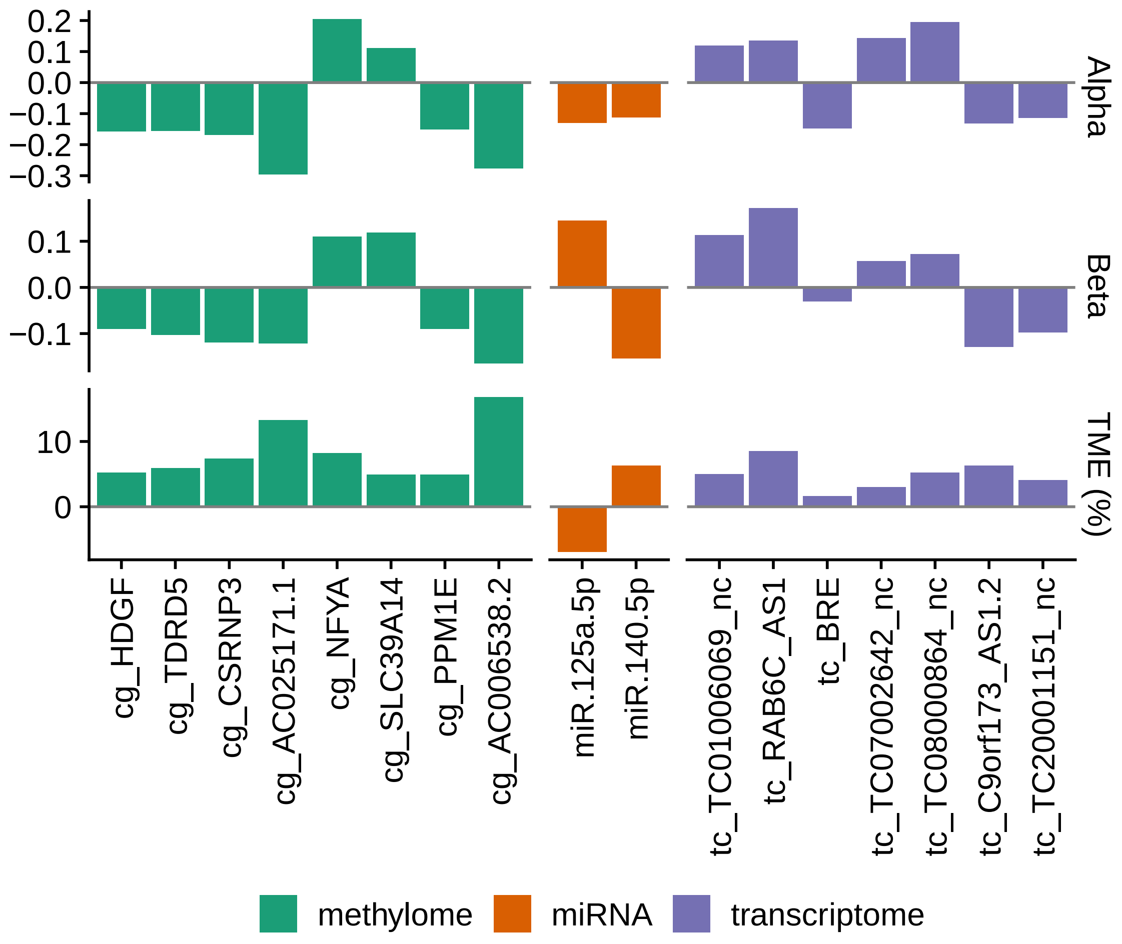


Figure 2.1: High dimensional mediation analysis with early integration and multiple omic layers identifies individual molecular features linking maternal mercury with childhood liver injury. Alpha represents the coefficient estimates of the exposure to the mediator, Beta indicates the coefficient estimates of the mediators to the outcome, and TME (%) represents the percent total effect mediated calculated as alpha*beta/gamma.

2.3 *Intermediate Integration*

High dimensional mediation with intermediate multiomic integration (Figure 1, panel b) identified differentially methylated CpG sites, gene transcript clusters, and miRNA which mediated associations of prenatal mercury with MAFLD risk in adolescents (Figure 2.2). Combining all omics layers before analysis identifies the strongest mediating feature across all omics layers without accounting for the differences in underlying correlation structure.

For this analysis, we use a novel two-step approach that incorporates feature level metadata to inform on feature selection using xtune (Zeng, Thomas, and Lewinger 2021), an approach that allows for feature-specific penalty parameters and, in this example, performs a group-lasso-type shrinkage within each omic dataset. This analysis is similar in theory to HIMA, with the difference being that for this method the penalization can vary across each of the omic layers. This analysis was based on the product of coefficients method for mediation and was performed in three steps:

1. ***Regression 1 (linear regression):***

First, we performed independent linear regression models for all exposure-mediator associations to get the exposure mediator coefficient:

$$m_i = a_0 + a_1 \times x \tag{2.1}$$

Where x is the exposure and m_i is each mediator.

2. ***Regression 2 (group lasso):***

Second, we performed a single group lasso regression for the mediator outcome associations, adjusting for the exposure, using the R package xtune (He and Zeng 2023). This step provided coefficients for each of the mediator outcome associations. We used bootstrapping to obtain the standard error of the coefficients from the group lasso regression for each of the mediator coefficients.

$$y = b_0 + b_1 \times x + b_{2_i} \times M \tag{2.2}$$

Where y is the outcome, x is the exposure, M is the mediator matrix with corresponding estimate b_{2_i}. The bootstrapped standard error (se) of b_{2_i} is used for calculating mediation confidence intervals.

3. ***Calculate mediation confidence interval for mediator (i):***

Finally, for each omic feature, we calculated the mediation effect and 95% confidence intervals using the R package RMediation, which is based on the distribution-of-the-product method (Tofighi and MacKinnon 2011).

$$\alpha = a_1 \tag{2.3}$$
$$\text{se of } \alpha = \text{se of } a_1 \tag{2.4}$$
$$\beta = b_{2_i} \tag{2.5}$$
$$\text{se of } \beta = \text{se of } b_{2_i} \tag{2.6}$$

Note: For the actual analysis, we would normally set n_boot to a higher value (1000 or more). In the example code, it is set to 12 to improve the speed of the function.

```
# Run Analysis
result_hima_intermediate <- hidimum(omics_lst = omics_lst,
                                    covs = covs,
                                    outcome = outcome,
                                    exposure = exposure,
                                    n_boot = 12,
                                    Y.family = "gaussian",
                                    integration = "intermediate")
```

```
# plot
plot_hidimum(result_hima_intermediate)
```

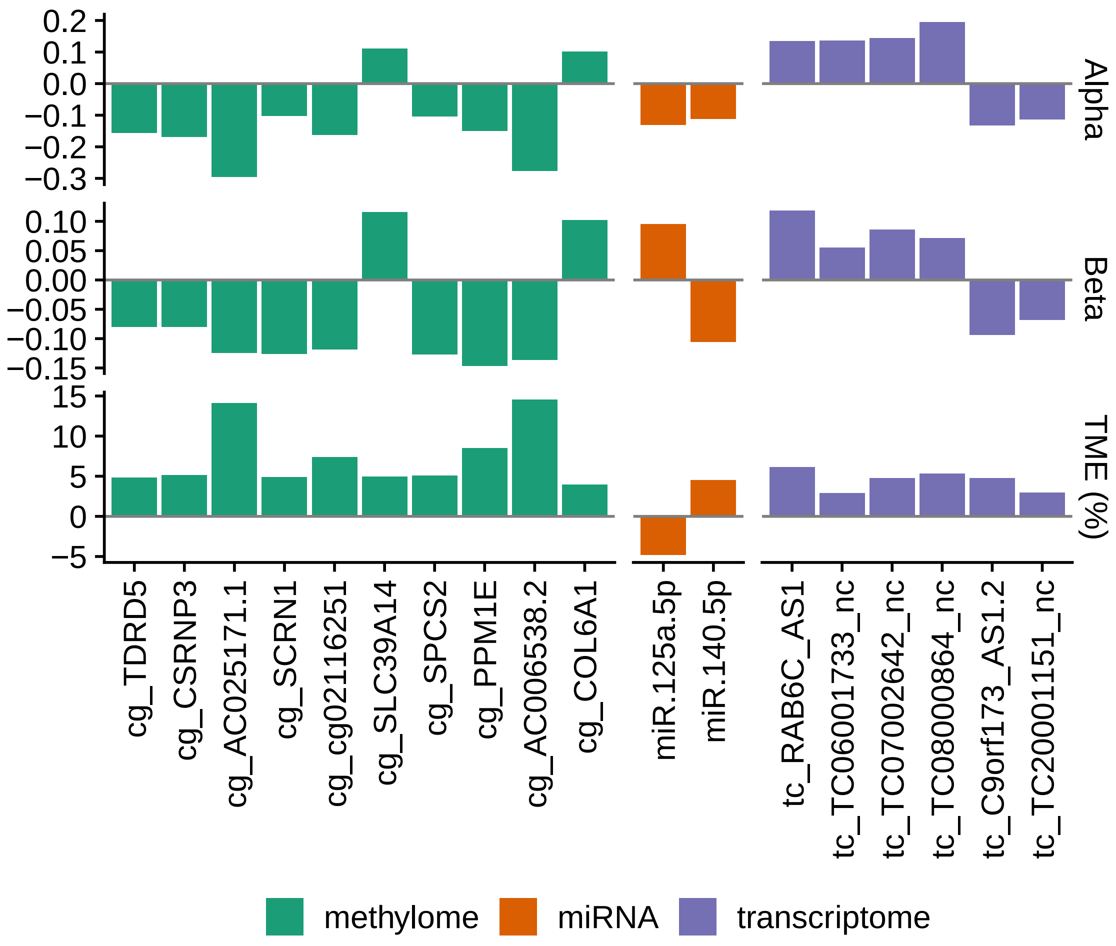


Figure 2.2: High dimensional mediation analysis with intermediate integration and multiple omic layers identifies individual molecular features linking maternal mercury with childhood liver injury. Alpha represents the coefficient estimates of the exposure to the mediator, Beta indicates the coefficient estimates of the mediators to the outcome, and TME (%) represents the percent total effect mediated calculated as alpha*beta/gamma.

2.4 *Late integration*

High dimensional mediation with late multiomic integration (Figure 1, panel c) identified differentially methylated CpG sites, gene transcript clusters, and miRNA which mediated associations of prenatal mercury with MAFLD risk in adolescents (Figure 2.3). High dimensional mediation with late multi-omic integration differs from the early and intermediate integration in that each omics layer is analyzed individually. Thus, this approach does not condition on features within the other omics layers in the analysis.

Combining all omics layers before analysis identifies the strongest mediating feature across all omics layers without accounting for the differences in underlying correlation structure. For this analysis, we used High Dimensional Mediation Analysis (HIMA), a penalization-based mediation method implemented in the R package HIMA (Zhang et al. 2016).

```r
# Run Analysis
result_hima_late <- hidimum(exposure = exposure,
                            outcome = outcome,
                            omics_lst = omics_lst,
                            covs = covs,
                            Y.family = "gaussian",
                            M.family = "gaussian",
                            integration = "late")

# plot
plot_hidimum(result_hima_late)
```

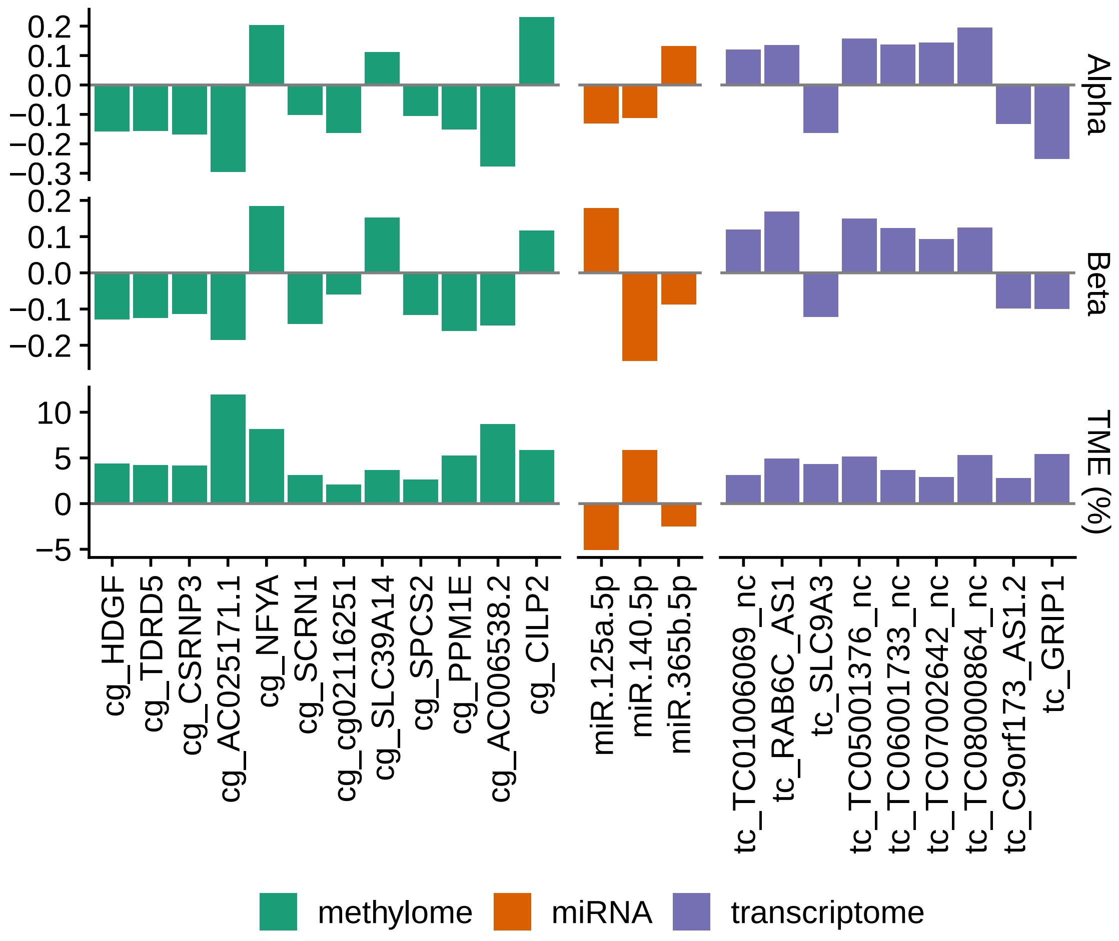


Figure 2.3: High dimensional mediation analysis with late integration and multiple omic layers identifies individual molecular features linking maternal mercury with childhood liver injury. Alpha represents the coefficient estimates of the exposure to the mediator, Beta indicates the coefficient estimates of the mediators to the outcome, and TME (%) represents the percent total effect mediated calculated as alpha*beta/gamma.

3 Mediation with latent factors

We define mediation with latent factors as a two step approach, in which we first perform dimensionality reduction on the omics data and then use the factors/clusters as latent mediator for the mediation analysis between the exposure and the outcome.

3.1 Load data and packages for Mediation Analysis with Latent factors

Before starting, load the data and packages using the following code.

```r
# Load R package
library(EnvirOmix)

# Load simulated data
data(simulated_data)

# Define exposure and outcome name
exposure <- simulated_data[["phenotype"]]$hs_hg_m_scaled
outcome  <- simulated_data[["phenotype"]]$ck18_scaled

# Get numeric matrix of covariates
covs <- simulated_data[["phenotype"]][c("e3_sex_None", "hs_child_age_yrs_None")]
covs$e3_sex_None <- ifelse(covs$e3_sex_None == "male", 1, 0)

# create list of omics data
omics_lst <- simulated_data[-which(names(simulated_data) == "phenotype")]

# Create data frame of omics data
omics_df <- omics_lst |>
  purrr::map(~as_tibble(.x, rownames = "name")) |>
  purrr::reduce(left_join, by = "name") |>
  column_to_rownames("name")
```

3.2 *Early integration*

For early integration, we used principal component analysis (PCA) as a dimensionality reduction step and selected the top i principal components which explained >80% of the variance. Following the joint dimensionality reduction step, we used the r package HIMA (Zhang et al. 2016) to examine whether the variance components mediated associations of in utero mercury exposure with MAFLD.

In this analysis, principal components explained >80% of the variance in the combined omics datasets. Of these components, 7 significantly mediated the relationship between maternal mercury and childhood liver injury (Figure 3.1).

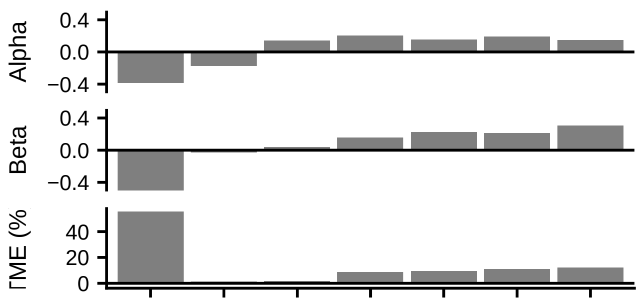
3.2.1 HIMA Early Integration

```r
# Run Analysis
result_med_with_latent_fctrs_early <-
  lafamum(exposure,
          outcome,
          omics_lst,
          covs = covs,
          Y.family = "gaussian",
          fdr.level = 0.05,
          integration = "early")
```

3.2.2 Plot Early Integration

```r
# Plot
(plot_lafamum(result_med_with_latent_fctrs_early))
```

a)



b)

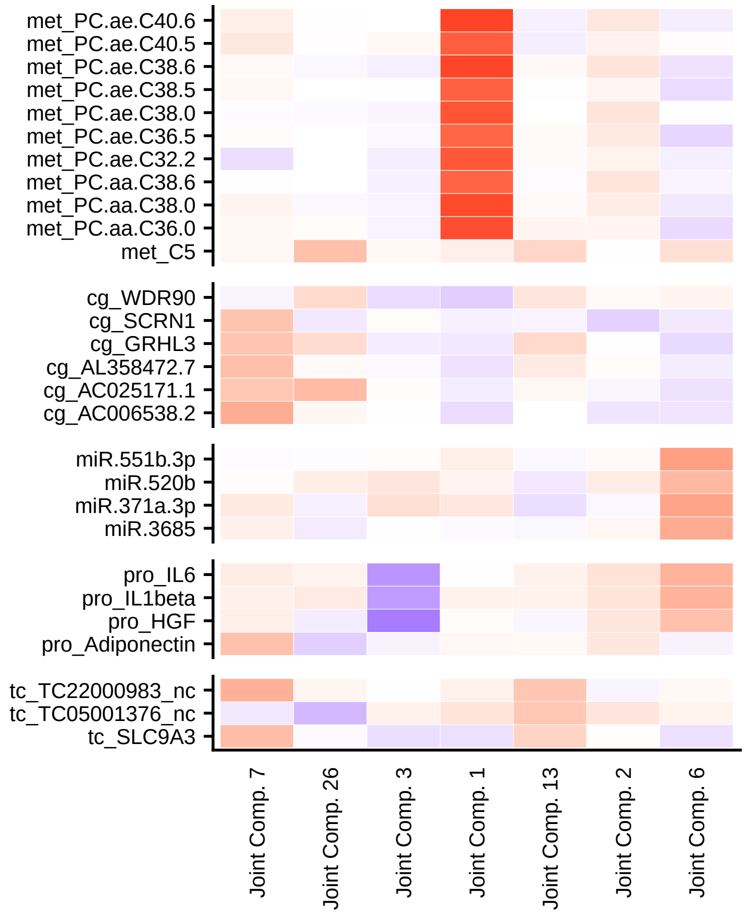


Figure 3.1: Mediation analysis with latent factors and early integration identifies joing components which mediate the association between maternal mercury and childhood liver injury. Panel A shows the mediation effects, where Alpha represents the coefficient estimates of the exposure to the mediator, Beta indicates the coefficient estimates of the mediators to the outcome, and TME (%) represents the percent total effect mediated calculated as alpha*beta/gamma. Panel B shows the individual correlation between the omic feature and the joint component.

3.3 *Intermediate Integration*

The steps for intermediate integration start with performing a joint dimensionality reduction step using Joint and Individual Variance Explained (JIVE) (Lock et al. 2013). Following the joint dimensionality reduction step, we used the r package HIMA (Zhang et al. 2016) to examine whether the variance components mediated associations of in utero mercury exposure with MAFLD.

3.3.1 Conduct JIVE and Perform Mediation Analysis

3.3.1.1 Conduct JIVE

For this step, JIVE can estimate the optimal number of joint and individual ranks by changing the `method` argument in the function `jive`. For the simulated HELIX data, the optimal number, determined by setting `method = "perm"`, was 22 joint ranks and 6, 9, 5, 5, and 8 ranks for the methylome, transcriptome, miRNA, proteome, and metabolome, respectively.

3.3.1.2 Perform mediation analysis

In this analysis, 6 joint components, 1 transcriptome specific component significantly mediated the relationship between maternal mercury and childhood liver injury (Figure 3.2).

```r
# Run analysis with rnkJ and rankA provided
result_med_with_latent_fctrs_JIVE <-
  lafamum(exposure, outcome,
          jive.rankJ = 22,
          jive.rankA = c(6, 9, 5, 5, 8),
          omics_lst,
          covs = covs,
          Y.family = "gaussian",
          fdr.level = 0.05,
          integration = "intermediate")
```

3.3.2 Plot Intermediate Integration

```r
(plot_lafamum(result_med_with_latent_fctrs_JIVE))
```

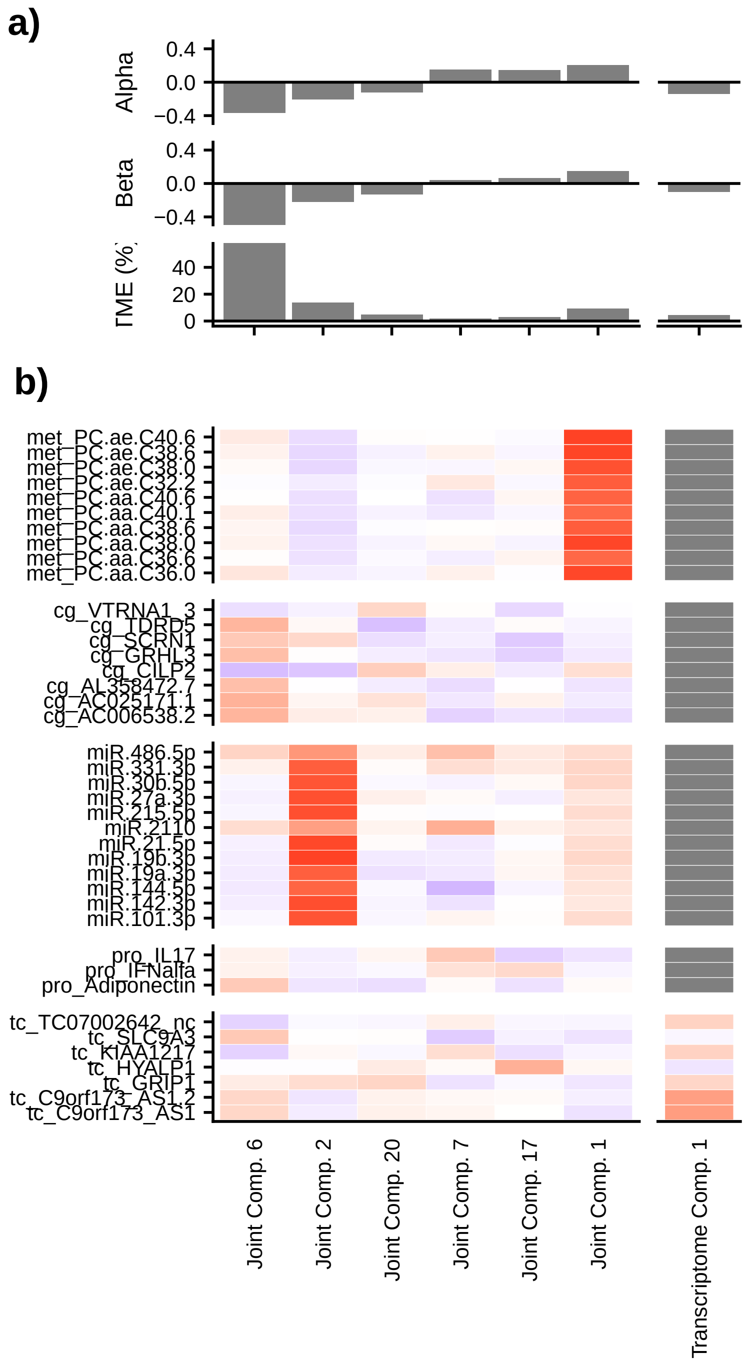


Figure 3.2: Mediation analysis with latent factors and intermediate integration identifies joint and individual variance componets which mediate the association between maternal mercury and childhood liver injury. Panel A shows the mediation effects, where Alpha represents the coefficient estimates of the exposure to the mediator, Beta indicates the coefficient estimates of the mediators to the outcome, and TME (%) represents the percent total effect mediated calculated as alpha*beta/gamma. Panel B shows the individual correlation between the omic feature and the joint and individual components.

3.4 *Late integration*

For late integration, we used principal component analysis (PCA) as a dimensionality reduction step on each omics layer separately, and selected the top i principal components which explained >80% of the variance. Following the dimensionality reduction step, we used the r package HIMA (Zhang et al. 2016) to examine whether the variance components mediated associations of in utero mercury exposure with MAFLD.

This analysis identified 2 methylated CpG sites, 1 miRNA, 1 protein and 2 expressed gene transcript clutesrs significantly mediated the association between mercury and MAFLD (Figure 3.3).

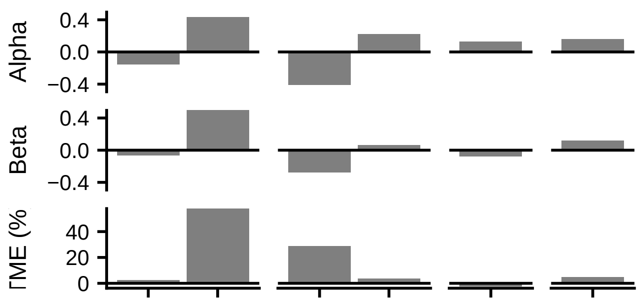
3.4.1 HIMA Late Integration

```
result_med_with_latent_fctrs_late <- lafamum(exposure,
                                             outcome,
                                             omics_lst,
                                             covs = covs,
                                             Y.family = "gaussian",
                                             fdr.level = 0.05,
                                             integration = "late")
```

3.4.2 Plot Late Integration

```
(plot_lafamum(result_med_with_latent_fctrs_late))
```

a)



b)

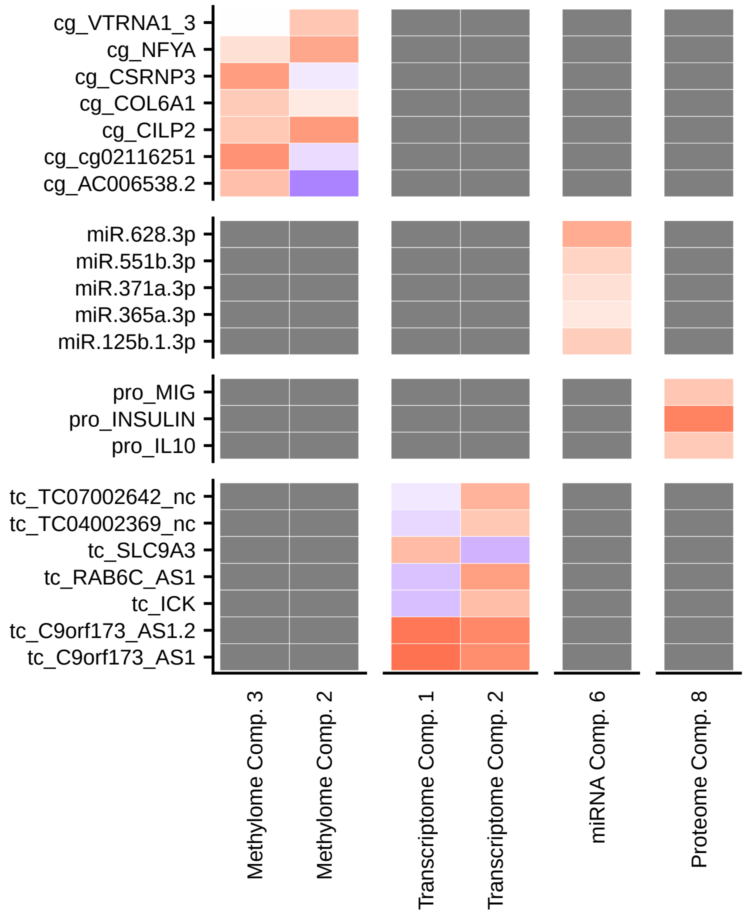


Figure 3.3: Mediation analysis with latent factors and late integration identifies features in each omics layer individually which mediates the association between maternal mercury and childhood liver injury. Panel A shows the mediation effects, where Alpha represents the coefficient estimates of the exposure to the mediator, Beta indicates the coefficient estimates of the mediators to the outcome, and TME (%) represents the percent total effect mediated calculated as alpha*beta/gamma. Panel B shows the individual correlation between the omic feature and components.

3.5 Pathway Analysis.

Following mediation analysis, you can use the correlation p-values to perform pathway analysis with appropriate pathway analysis software.

4 Integrated/Quasi-Mediation

4.1 Load data and packages

Before starting, load required packages. The code in this section relies heavily on the functions from the R package LUCIDus (Peng et al. 2020).

```r
# General Packages:
library(EnvirOmix)
library(tidyverse)
library(table1)
# Load plotting packages:
library(ggplot2)
library(cowplot)
library(ComplexHeatmap)
library(ggh4x)
# Packages for Quasi-mediation:
#LUCIDus Version 3.0.1 is required for this analysis:
library(LUCIDus)
library(networkD3)
library(plotly)
library(htmlwidgets)
library(jsonlite)
```

And load the data:

```r
# Load R package
library(EnvirOmix)

# Load simulated data
data(simulated_data)

# Define exposure and outcome name
exposure <- simulated_data[["phenotype"]]$hs_hg_m_scaled
outcome  <- simulated_data[["phenotype"]]$ck18_scaled

# Get numeric matrix of covariates
covs <- simulated_data[["phenotype"]][c("e3_sex_None", "hs_child_age_yrs_None")]
covs$e3_sex_None <- ifelse(covs$e3_sex_None == "male", 1, 0)
```

```r
# create list of omics data
omics_lst <- simulated_data[-which(names(simulated_data) == "phenotype")]

# Create data frame of omics data
omics_df <- omics_lst |>
  purrr::map(~as_tibble(.x, rownames = "name")) |>
  purrr::reduce(left_join, by = "name") |>
  column_to_rownames("name")
```

For this analysis, we will subset to only three omics layers to reduce the computational burden.

```r
# Three omics layers, Methylation (CpG), Transcriptome and miRNA
omics_df_analysis <- omics_df %>%
  dplyr::select(contains("cg"), contains("TC"), contains("miR"))
```

The figures for quasi-mediation rely on several custom functions. The code for functions are provided in Chapter 6.

4.2 *Early Integration*

In the Early Integration model, genomic/exposomic exposures G, other omics data Z and phenotype trait Y are integrated through a latent categorical variable X, as implemented in the R package LUCIDus (Peng et al. 2020). Because X is an unobserved categorical variable, each category of X is interpreted as a latent cluster in the data, jointly defined by G, Z and Y. Let G be a $N \times P$ matrix with columns representing genetic/environmental exposures, and rows representing the observations; Z be a $N \times M$ matrix of omics data (for example, gene expression data, DNA methylation profiles and metabolomic data etc.) and Y be a N-length vector of phenotype trait. We further assume G, Z and Y are measured through a prospective sampling procedure so we do not model the distribution of G. All three measured components (G, Z and Y) are linked by a latent variable X consisting of K categories. The distributions of X given G, Z given X and Y given X are conditionally independent with each other. Let $f(\cdot)$ denote the probability mass functions (PMF) for categorical random variables or the probability density functions (PDF) for continuous random variables. The joint log-likelihood of the LUCID model is constructed as:

$$\begin{aligned}\log L(\Theta) &= \sum_{i=1}^{N} \log f(Z_i, Y_i | G_i; \Theta) \\ &= \sum_{i=1}^{N} \log \sum_{j=1}^{K} f(X_i = j | G_i; \Theta) f(Z_i | X_i = j; \Theta) f(Y_i | X_i = j; \Theta)\end{aligned}$$

where Θ is a generic notation for all parameters in Early Integration LUCID model. EM algorithm is implemented to estimate all parameters Θ iteratively until convergence, and Θ represents the G to X, X to Z, and X to Y associations,

We a-priori assigned the number of clusters to two. Early Integration LUCID estimates two omics specific clusters which represent an differential risks for the outcome. These omic profiles confer different risks of the outcome and represent each omics feature's contribution to the exposure-outcome association.

4.2.1 Analysis: Early Integration

When using `estimate_lucid()` to fit the Early Integration LUCID model, we specify `lucid_model = "early"`, and `K` is an integer representing number of latent clusters. `G`, `Z`, `Y` are the inputs for exposure, omics data matrix, and the outcome, respectively.`CoY` are the covariates to be adjusted for the X to Y association and `CoG` are the covariates to be adjusted for the G to X association. We specify `useY = TRUE` to construct supervised LUCID model. Otherwise, `useY = FALSE` will construct unsupervised LUCID model. `init_par = "random"` means that we initiate the parameters with random guess. We specify `family = "normal"` since the outcome is continuous. If the outcome is binary, we would specify `family = "binary"`.The full code for the `sankey_early_integration` function is provided in Section 6.1.1

```r
G = exposure |> as.matrix()
Z = omics_df_analysis |> as.matrix()

fit1 <- estimate_lucid(lucid_model = "early",
                       G = G,
                       Z = Z,
                       Y = outcome,
                       K = 2,
                       CoY = covs,
                       CoG = covs,
                       useY = TRUE,
                       init_par = "random",
                       family = "normal")
## .....
```

4.2.2 Sankey Diagram

```r
p1 <- sankey_early_integration(fit1, text_size = 20)
```

4.2.3 Early Integration Results

Omics profiles for each cluster determined using Early Integration LUCID. The full code for the `plot_omics_profiles` function is provided in Section 6.1.4

```r
plot_omics_profiles(fit1, "Early", omics_lst)
```

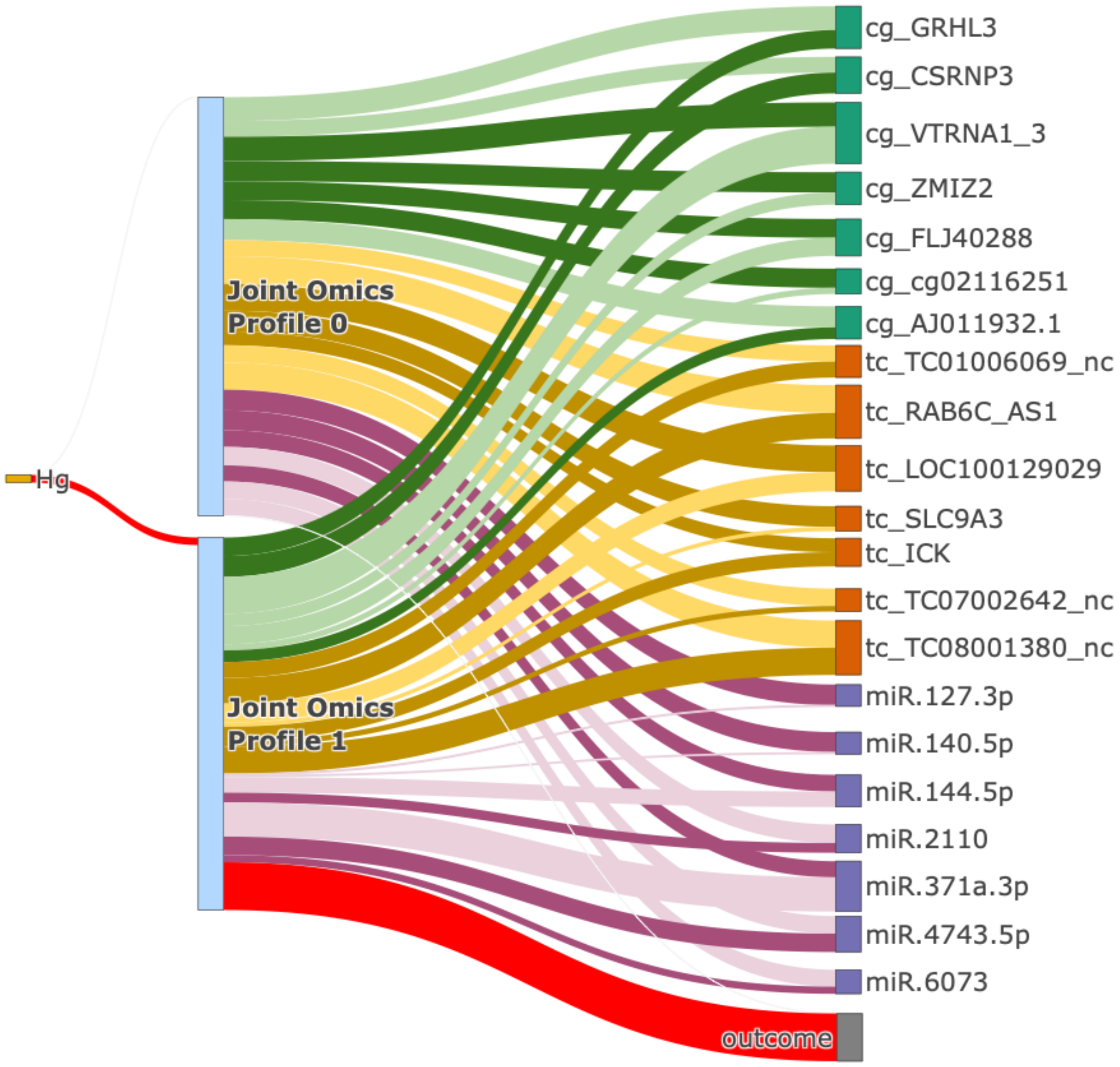
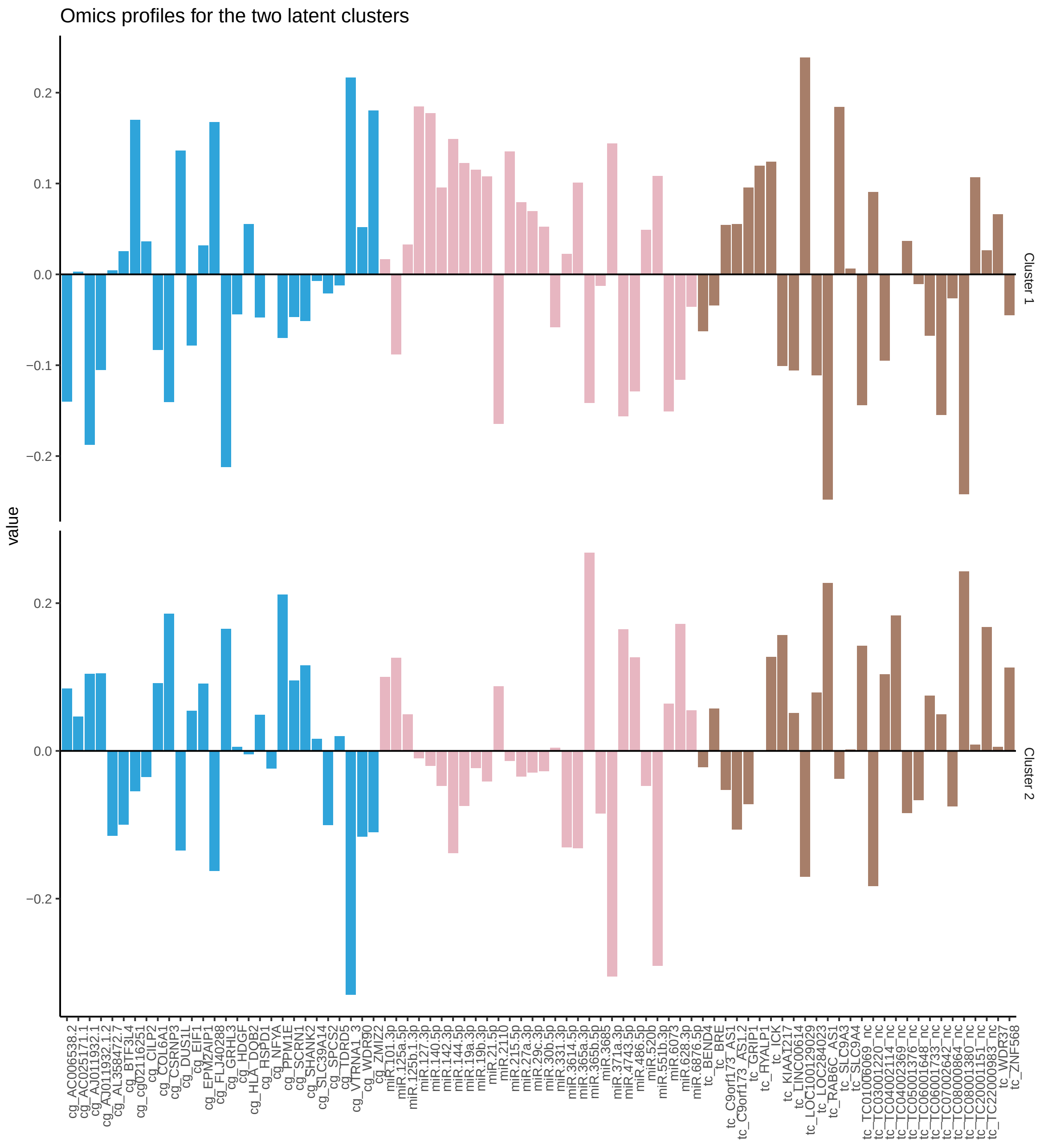


Figure 4.1: The Sankey Diagram for LUCID (Early Integration).



4.3 *Intermediate Integration*

In LUCID in parallel conducting intermediate integration, latent clusters X_a are estimated in each omics layer separately while integrating information from the genetic/environmental exposure G and the outcome Y by assuming no correlations across different omics layers. Let G be a $N \times P$ matrix with columns representing genetic/environmental exposures, and rows representing the observations; there is a collection of m omics data, denoted by $Z_1, \ . \ . \ . \ , Z_a, \ . \ . \ . \ , Z_m$ with corresponding dimensions $p_1, \ . \ . \ . \ , p_a,$

. . . , p_m. Each omics data Z_a is summarized by a latent categorical variable X_a, which contains K_a categories. Each category is interpreted as a latent cluster (or subgroup) for that particular omics layer and Y be a N-length vector of phenotype trait. Let D be the generic notation for all observed data. The log likelihood of LUCID with multiple latent variables is constructed below,

$$
\begin{aligned}
l(\Theta \mid D) &= \sum_{i=1}^{n} \log f\left(Z_{1i}, \ldots, Z_{mi}, Y_i \mid G_i; \Theta\right) \\
&= \sum_{i=1}^{n} \log \left[\prod_{j_1=1}^{k_1} \cdots \prod_{j_m=1}^{k_m} f\left(Z_{1i}, \ldots, Z_{mi}, X_{1i}, \ldots, X_{mi}, Y_i \mid G_i; \Theta\right)^{I(X_{1i}=j_1, \ldots, X_{mi}=j_m)} \right] \\
&= \sum_{i=1}^{n} \sum_{j_1=1}^{k_1} \cdots \sum_{j_m=1}^{k_m} I\left(X_{1i}=j_1, \ldots, X_{mi}=j_m\right) \log f\left(Z_{1i}, \ldots, Z_{mi}, X_{1i}, \ldots, X_{mi}, Y_i \mid G_i; \Theta\right) \\
&= \sum_{i=1}^{n} \sum_{j_1=1}^{k_1} \cdots \sum_{j_m=1}^{k_m} I\left(X_{1i}=j_1, \ldots, X_{mi}=j_m\right) \log \phi\left(Y_i \mid X_{1i}, \ldots, X_{mi}, \delta, \sigma^2\right) \\
&+ \sum_{i=1}^{n} \sum_{a=1}^{m} \sum_{j_1=1}^{k_1} \cdots \sum_{j_m=1}^{k_m} I\left(X_{1i}=j_1, \ldots, X_{mi}=j_m\right) \log \phi\left(Z_{ai} \mid X_{ai}=j_a, \mu_{a,j_a}, \Sigma_{a,j_a}\right) \\
&+ \sum_{i=1}^{n} \sum_{a=1}^{m} \sum_{j_1=1}^{k_1} \cdots \sum_{j_m=1}^{k_m} I\left(X_{1i}=j_1, \ldots, X_{mi}=j_m\right) \log S\left(X_{ai}=j_a \mid G_i, \beta_a\right)
\end{aligned}
$$

The log likelihood of LUCID in parallel is similar to that with Early integration LUCID. It is natural to follow the same principles of EM algorithm for Early integration LUCID with single intermediate variable.

We a-priori assigned the number of clusters for each omic layer to two. LUCID in parallel estimates omics specific clusters which represent an differential risks for the outcome within the layer. For each set of latent clusters within each omics layer, the corresponding omics profile was computed to identify the independent contribution of each omics layer to the exposure-outcome association.

4.3.1 Analysis: LUCID with 3 omics layers in parallel

When using `estimate_lucid()` to fit LUCID in parallel model, we specify `lucid_model = "parallel"`, and `K` is a list with the same length as the number of omics layers and each element in the list is an integer representing number of latent clusters for the corresponding omics layer. `max_itr = 200` means that the algorithm will stop when the iterations reaches 200 if still not reaching convergence. We specify `useY = TRUE` to construct supervised LUCID model. `modelName` is a vector of strings specifies the geometric model of omics data, the default `modelName` is "VVV", but here "EEV" is more suitable.

```r
G = exposure %>% as.matrix()
Z = omics_lst[c(1:3)]
Y = outcome

fit <- estimate_lucid(lucid_model = "parallel",
                      G = G,
                      Z = Z,
                      Y = Y,
                      K = rep(2, length(Z)),
                      family = "normal",
                      max_itr = 200,
                      useY = TRUE,
                      init_omic.data.model  = "EEV")

# Reorder the clusters
fit_reordered <- reorder_lucid_parallel(fit, reference = c(2,2,2))
```

4.3.2 Sankey Diagram

The full code for the `plot_lucid_in_parellel_plotly` function is provided in Section 6.1.2

```
p2 <- plot_lucid_in_parallel_plotly(fit_reordered,
                                    sankey_colors = sankey_colors,
                                    text_size = 20,
                                    n_z_ftrs_to_plot = c(7,7,7))
```

4.3.3 Omics profiles for each cluster predicted by LUCID

The full code for the `plot_omics_profiles` function is provided in Section 6.1.4

```
plot_omics_profiles(fit_reordered, "Intermediate", omics_lst)
```

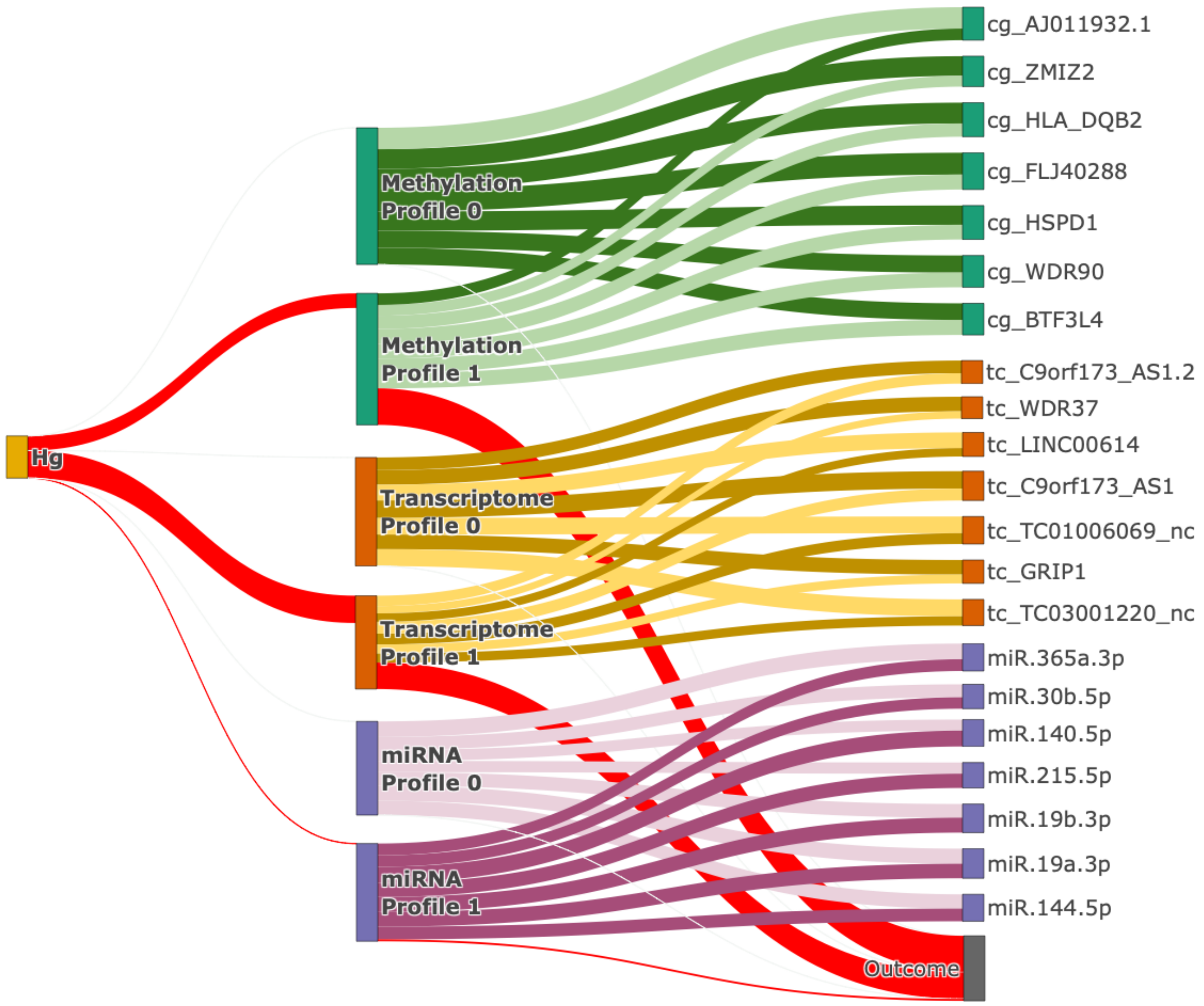
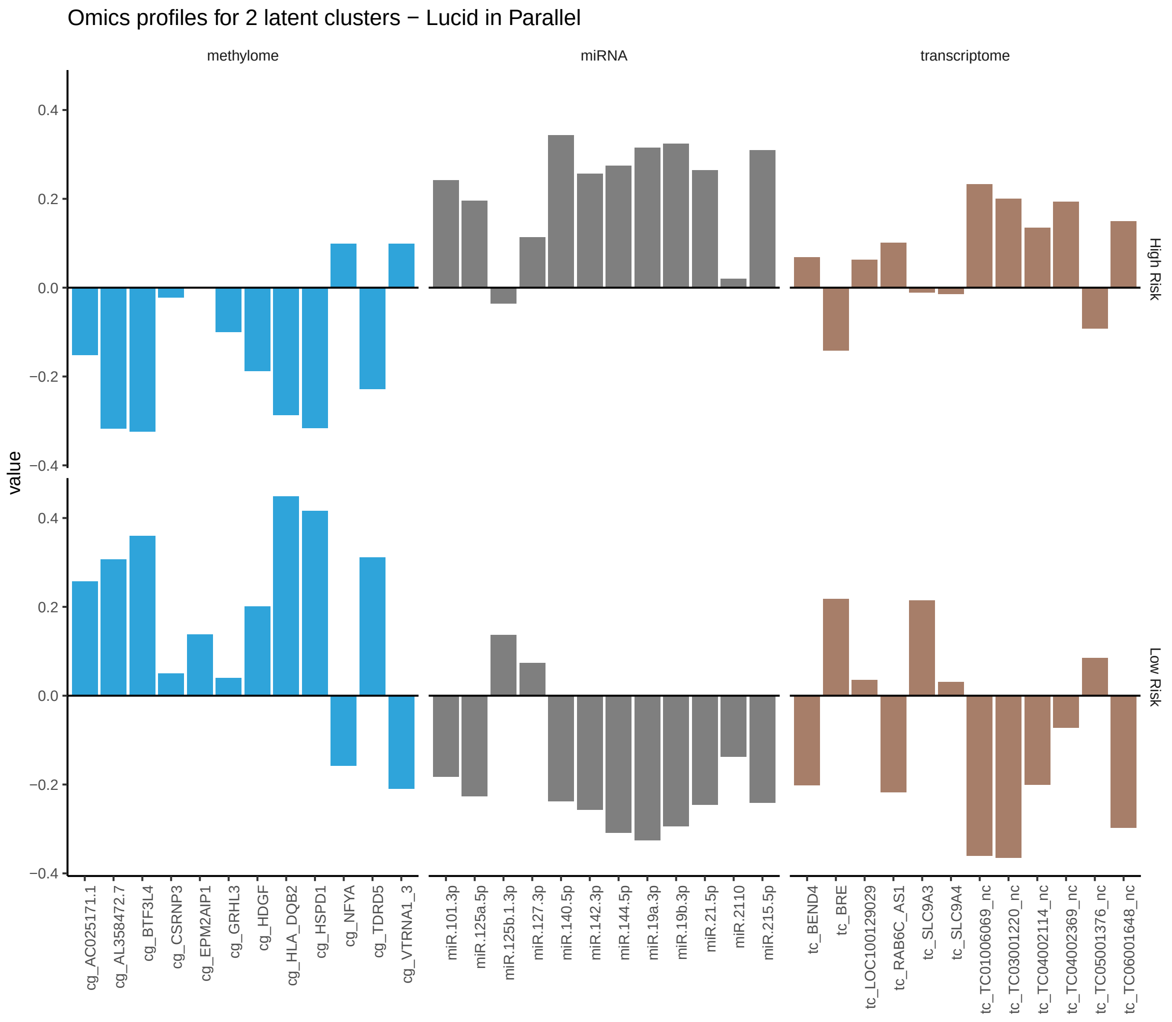


Figure 4.2: The Sankey Diagram for LUCID in Parallel (Intermediate Integration).



4.4 *Late Integration*

For late integration, LUCID in serial is implemented, which successively links multiple single omics LUCID models using the Early Integration LUCID model. Let G be a $N \times P$ matrix with columns representing genetic/environmental exposures, and rows representing the observations; there is an ordered collection of m omics data, denoted by $Z_1, \ldots, Z_a, \ldots, Z_m$ with corresponding dimensions $p_1, \ldots, p_a, \ldots, p_m$. Successive omics layers Z_a are linked by using each observations' posterior inclusion probability (PIP) for latent clusters X_a in the initial LUCID model to be the "exposure" variable for each successive model. The omics layers are ordered in a sequential fashion based on the biological relationships between omics layers. For the first model, an unsupervised Early Integration LUCID model using G as the exposure and Z_1 as the omics layer. The PIPs of the non-reference clusters were extracted and used as the input for the exposure for the following unsupervised Early Integration LUCID model. This procedure is iterated until the last omics layer Z_m, for which a supervised Early Integration LUCID model is used to conduct integrated clustering while using PIPs from the previous LUCID model and information on the outcome Y.

4.4.1 Analysis: LUCID with 3 omics layers in serial

When using `estimate_lucid()` to fit LUCID in serial model, we specify `lucid_model = "serial"`, and `K` is a list with the same length as the number of ordered omics layers and each element in the list is an integer representing number of latent clusters for the corresponding omics layer. `G`, `Z`, `Y` are the inputs for exposure, omics data matrix, and the outcome, respectively. Note that here Z is list of omics layers (vectors). `CoY` are the covariates to be adjusted for the X to Y association and `CoG` are the covariates to be adjusted for the G to X association. We specify `useY = TRUE` to construct supervised LUCID in serial model. We specify `family = "normal"` since the outcome is continuous. If the outcome is binary, we would specify `family = "binary"`. We specify `Rho_Z_Mu = 10` and `Rho_Z_Cov = .3` to use LASSO penalty to regularize cluster-specific means and variance for Z. We will get a selection of Z features for each layer, and then we refit the LUCID in serial model with only selected Z features to construct the final model.

```r
set.seed(100)
G_Hg = exposure %>% as.matrix()
Z = list(omics_lst$methylome,
         omics_lst$transcriptome,
         omics_lst$miRNA)
Y_liv_inj = scale(outcome)

# Run the LUCID Model
fit <- estimate_lucid(lucid_model = "serial",
                      G = G_Hg,
                      Z = Z,
                      Y = Y_liv_inj,
                      CoY = covs,
                      CoG = covs,
                      K = list(2,2,2),
                      useY = TRUE,
                      family = "normal",
                      Rho_Z_Mu = 10,
                      Rho_Z_Cov = .3)

#extract selected Z features to to refit lUCID in serial
selected_Z = vector(mode = "list", length = length(Z))
for (i in (1:length(Z))){
  fiti_select_Z = fit$submodel[[i]]$select$selectZ
  selected_Z[[i]] = Z[[i]][,fiti_select_Z]
}

#Refit ther LUCID in serial model with selected Z
fit <- estimate_lucid(lucid_model = "serial",
                      G = G_Hg,
                      Z = selected_Z,
                      Y = Y_liv_inj,
                      CoY = covs,
                      CoG = covs,
                      K = list(2,2,2),
                      useY = TRUE,
                      init_par = "random",
                      family = "normal")

# Rename Exposure
fit1 <- fit$submodel[[1]]
fit1$var.names$Gnames[1] <- "Hg"
fit2 <- fit$submodel[[2]]
fit2$var.names$Gnames[1] <- "<b>Methylation\nProfile 1</b>"
fit3 <- fit$submodel[[3]]
fit3$var.names$Gnames[1] <- "<b>miRNA\nProfile 1</b>"
```

4.4.2 Sankey Diagram

The full code for the `sankey_in_serial` function is provided in Section 6.1.3.

```r
col_pal <- RColorBrewer::brewer.pal(n = 8, name = "Dark2")
color_pal_sankey <- matrix(c("exposure", "red",
                             "lc"      , "#b3d8ff",
                             "TC"      , col_pal[2],
                             "CpG"     , col_pal[1],
                             "miRNA"   , col_pal[3],
                             "outcome" , "grey"),
                           ncol = 2, byrow = TRUE) %>%
  as_tibble(.name_repair = "unique") %>%
  janitor::clean_names() %>%
  dplyr::rename(group = x1, color = x2)

p3<- sankey_in_serial(fit1,
                      fit2,
                      fit3,
                      color_pal_sankey,
                      text_size = 24)
```

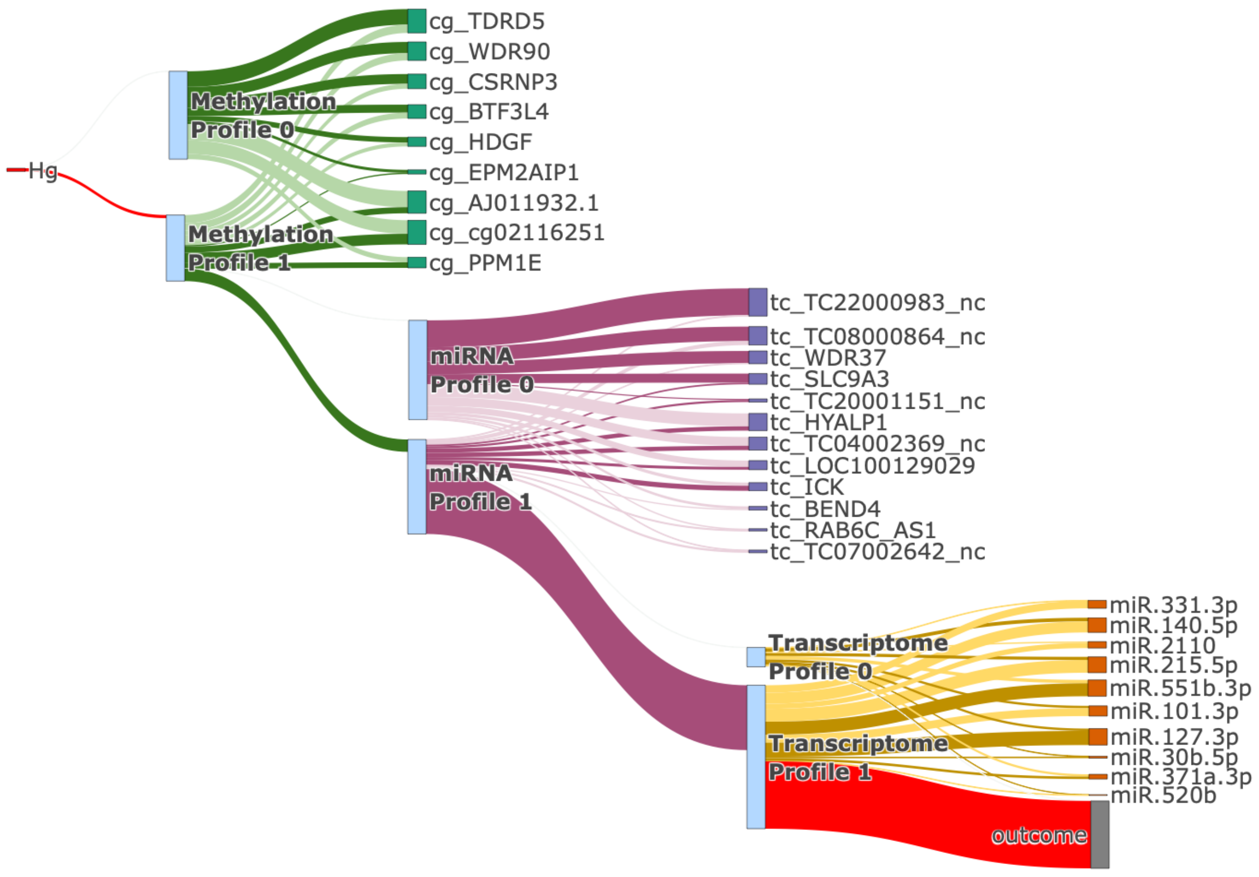


Figure 4.3: The Sankey Diagram for LUCID in Serial (Late Integration).

5 Software

5.1 *Installing R packages*

The following code will download all the packages needed to run the analyses in this book.

```r
# Download EnvirOmix package if not already installed:
if(!requireNamespace("EnvirOmix", quietly = TRUE)){
  if(!requireNamespace("devtools", quietly = TRUE)) install.packages("devtools")
  devtools::install_github("Goodrich-Lab/EnvirOmix")
}

# List all CRAN packages used in this book:
cran_packages <- c("tidyverse",
                   "tools",
                   "parallel",
                   "boot",
                   "table1",
                   "BiocManager",
                   "ggplot2",
                   "cowplot",
                   "ggh4x",
                   "LUCIDus",
                   "networkD3",
                   "plotly",
                   "htmlwidgets",
                   "jsonlite")

# Install cran packages if not already installed:
for(package_name in cran_packages){
  if(!requireNamespace(package_name, quietly = TRUE)){
    install.packages(package_name, repos = "http://cran.r-project.org/")
  }
}

# List all Bioconductor packages used in this book:
bioconductor_packages <- c("BiocGenerics",
                           "ComplexHeatmap",
                           "IRanges",
                           "S4Vectors")
```

```r
# Install bioconductor packages if not already installed:
for(package_name in bioconductor_packages){
  if(!requireNamespace(package_name, quietly = TRUE)){
    BiocManager::install(package_name)
  }
}
```

5.2 *R development environment*

The following table lists the package versions and corresponding repository of all the R packages that were used to develop the code in this book.

Table 5.1: List of all R Packages used in this book.

Package	Version	Attached or Loaded	Date/Publication	Source
ComplexHeatmap	2.16.0	Attached	2023-05-08	Bioconductor
cowplot	1.1.1	Attached	2020-12-30	CRAN (R 4.3.0)
devtools	2.4.5	Attached	2022-10-11	CRAN (R 4.3.0)
dplyr	1.1.3	Attached	2023-09-03	CRAN (R 4.3.0)
EnvirOmix	0.0.0.9000	Attached	2023-11-03	Bioconductor
forcats	1.0.0	Attached	2023-01-29	CRAN (R 4.3.0)
ggh4x	0.2.6	Attached	2023-08-30	CRAN (R 4.3.0)
ggplot2	3.4.4	Attached	2023-10-12	CRAN (R 4.3.1)
htmlwidgets	1.6.2	Attached	2023-03-17	CRAN (R 4.3.0)
jsonlite	1.8.7	Attached	2023-06-29	CRAN (R 4.3.0)
lubridate	1.9.3	Attached	2023-09-27	CRAN (R 4.3.1)
LUCIDus	3.0.1	Attached	2023-10-31	CRAN (R 4.3.1)
networkD3	0.4	Attached	2017-03-18	CRAN (R 4.3.0)
plotly	4.10.2	Attached	2023-06-03	CRAN (R 4.3.0)
purrr	1.0.2	Attached	2023-08-10	CRAN (R 4.3.0)
readr	2.1.4	Attached	2023-02-10	CRAN (R 4.3.0)
stringr	1.5.0	Attached	2022-12-02	CRAN (R 4.3.0)
table1	1.4.3	Attached	2023-01-06	CRAN (R 4.3.0)
tibble	3.2.1	Attached	2023-03-20	CRAN (R 4.3.0)
tidyr	1.3.0	Attached	2023-01-24	CRAN (R 4.3.0)
tidyverse	2.0.0	Attached	2023-02-22	CRAN (R 4.3.0)
usethis	2.2.2	Attached	2023-07-06	CRAN (R 4.3.0)
abind	1.4-5	Loaded	2016-07-21	CRAN (R 4.3.0)
adaptMCMC	1.4	Loaded	2021-03-29	CRAN (R 4.3.0)
arm	1.13-1	Loaded	2022-08-28	CRAN (R 4.3.0)
backports	1.4.1	Loaded	2021-12-13	CRAN (R 4.3.0)
BiocGenerics	0.46.0	Loaded	2023-06-04	Bioconductor
BiocManager	1.30.22	Loaded	2023-08-08	CRAN (R 4.3.0)
bitops	1.0-7	Loaded	2021-04-24	CRAN (R 4.3.0)
boot	1.3-28.1	Loaded	2022-11-22	CRAN (R 4.3.1)
broom	1.0.5	Loaded	2023-06-09	CRAN (R 4.3.0)
cachem	1.0.8	Loaded	2023-05-01	CRAN (R 4.3.0)

Table 5.1: List of all R Packages used in this book. *(continued)*

Package	Version	Attached or Loaded	Date/Publication	Source
callr	3.7.3	Loaded	2022-11-02	CRAN (R 4.3.0)
caTools	1.18.2	Loaded	2021-03-28	CRAN (R 4.3.0)
circlize	0.4.15	Loaded	2022-05-10	CRAN (R 4.3.0)
class	7.3-22	Loaded	2023-05-03	CRAN (R 4.3.1)
cli	3.6.1	Loaded	2023-03-23	CRAN (R 4.3.0)
clue	0.3-65	Loaded	2023-09-23	CRAN (R 4.3.1)
cluster	2.1.4	Loaded	2022-08-22	CRAN (R 4.3.1)
coda	0.19-4	Loaded	2020-09-30	CRAN (R 4.3.0)
codetools	0.2-19	Loaded	2023-02-01	CRAN (R 4.3.1)
colorspace	2.1-0	Loaded	2023-01-23	CRAN (R 4.3.0)
conquer	1.3.3	Loaded	2023-03-06	CRAN (R 4.3.0)
crayon	1.5.2	Loaded	2022-09-29	CRAN (R 4.3.0)
data.table	1.14.8	Loaded	2023-02-17	CRAN (R 4.3.0)
digest	0.6.33	Loaded	2023-07-07	CRAN (R 4.3.0)
doParallel	1.0.17	Loaded	2022-02-07	CRAN (R 4.3.0)
e1071	1.7-13	Loaded	2023-02-01	CRAN (R 4.3.0)
ellipsis	0.3.2	Loaded	2021-04-29	CRAN (R 4.3.0)
epiomics	1.0.0	Loaded	2023-03-15	CRAN (R 4.3.0)
evaluate	0.23	Loaded	2023-11-01	CRAN (R 4.3.1)
fansi	1.0.5	Loaded	2023-10-08	CRAN (R 4.3.1)
fastmap	1.1.1	Loaded	2023-02-24	CRAN (R 4.3.0)
fdrtool	1.2.17	Loaded	2021-11-13	CRAN (R 4.3.0)
foreach	1.5.2	Loaded	2022-02-02	CRAN (R 4.3.0)
Formula	1.2-5	Loaded	2023-02-24	CRAN (R 4.3.0)
fs	1.6.3	Loaded	2023-07-20	CRAN (R 4.3.0)
generics	0.1.3	Loaded	2022-07-05	CRAN (R 4.3.0)
GetoptLong	1.0.5	Loaded	2020-12-15	CRAN (R 4.3.0)
ggrepel	0.9.4	Loaded	2023-10-13	CRAN (R 4.3.1)
glasso	1.11	Loaded	2019-10-01	CRAN (R 4.3.0)
glmnet	4.1-8	Loaded	2023-08-22	CRAN (R 4.3.0)
GlobalOptions	0.1.2	Loaded	2020-06-10	CRAN (R 4.3.0)
glue	1.6.2	Loaded	2022-02-24	CRAN (R 4.3.0)
gplots	3.1.3	Loaded	2022-04-25	CRAN (R 4.3.0)
gridExtra	2.3	Loaded	2017-09-09	CRAN (R 4.3.0)
gtable	0.3.4	Loaded	2023-08-21	CRAN (R 4.3.0)
gtools	3.9.4	Loaded	2022-11-27	CRAN (R 4.3.0)
hdi	0.1-9	Loaded	2021-05-27	CRAN (R 4.3.0)
HDMT	1.0.5	Loaded	2022-01-29	CRAN (R 4.3.0)
here	1.0.1	Loaded	2020-12-13	CRAN (R 4.3.0)
HIMA	2.2.1	Loaded	2023-09-10	CRAN (R 4.3.0)
hms	1.1.3	Loaded	2023-03-21	CRAN (R 4.3.0)
hommel	1.6	Loaded	2021-12-17	CRAN (R 4.3.0)
htmltools	0.5.6.1	Loaded	2023-10-06	CRAN (R 4.3.1)
httpuv	1.6.11	Loaded	2023-05-11	CRAN (R 4.3.0)
httr	1.4.7	Loaded	2023-08-15	CRAN (R 4.3.0)
igraph	1.5.1	Loaded	2023-08-10	CRAN (R 4.3.0)
intervals	0.15.4	Loaded	2023-06-29	CRAN (R 4.3.0)

Table 5.1: List of all R Packages used in this book. *(continued)*

Package	Version	Attached or Loaded	Date/Publication	Source
IRanges	2.34.1	Loaded	2023-07-02	Bioconductor
iterators	1.0.14	Loaded	2022-02-05	CRAN (R 4.3.0)
KernSmooth	2.23-22	Loaded	2023-07-10	CRAN (R 4.3.0)
knitr	1.44	Loaded	2023-09-11	CRAN (R 4.3.0)
lars	1.3	Loaded	2022-04-13	CRAN (R 4.3.0)
later	1.3.1	Loaded	2023-05-02	CRAN (R 4.3.0)
lattice	0.21-9	Loaded	2023-10-01	CRAN (R 4.3.1)
lavaan	0.6-16	Loaded	2023-07-19	CRAN (R 4.3.0)
lazyeval	0.2.2	Loaded	2019-03-15	CRAN (R 4.3.0)
lbfgs	1.2.1.2	Loaded	2022-06-23	CRAN (R 4.3.0)
lifecycle	1.0.3	Loaded	2022-10-07	CRAN (R 4.3.0)
linprog	0.9-4	Loaded	2022-03-09	CRAN (R 4.3.0)
lme4	1.1-34	Loaded	2023-07-04	CRAN (R 4.3.0)
lmerTest	3.1-3	Loaded	2020-10-23	CRAN (R 4.3.0)
lpSolve	5.6.19	Loaded	2023-09-13	CRAN (R 4.3.0)
magrittr	2.0.3	Loaded	2022-03-30	CRAN (R 4.3.0)
MASS	7.3-60	Loaded	2023-05-04	CRAN (R 4.3.1)
Matrix	1.6-1.1	Loaded	2023-09-18	CRAN (R 4.3.1)
MatrixModels	0.5-2	Loaded	2023-07-10	CRAN (R 4.3.0)
matrixStats	1.0.0	Loaded	2023-06-02	CRAN (R 4.3.0)
mclust	6.0.0	Loaded	2022-10-31	CRAN (R 4.3.0)
memoise	2.0.1	Loaded	2021-11-26	CRAN (R 4.3.0)
mime	0.12	Loaded	2021-09-28	CRAN (R 4.3.0)
miniUI	0.1.1.1	Loaded	2018-05-18	CRAN (R 4.3.0)
minqa	1.2.6	Loaded	2023-09-11	CRAN (R 4.3.0)
mnormt	2.1.1	Loaded	2022-09-26	CRAN (R 4.3.0)
modelr	0.1.11	Loaded	2023-03-22	CRAN (R 4.3.0)
munsell	0.5.0	Loaded	2018-06-12	CRAN (R 4.3.0)
ncvreg	3.14.1	Loaded	2023-04-25	CRAN (R 4.3.0)
nlme	3.1-163	Loaded	2023-08-09	CRAN (R 4.3.0)
nloptr	2.0.3	Loaded	2022-05-26	CRAN (R 4.3.0)
nnet	7.3-19	Loaded	2023-05-03	CRAN (R 4.3.1)
numDeriv	2016.8-1.1	Loaded	2019-06-06	CRAN (R 4.3.0)
OpenMx	2.21.8	Loaded	2023-04-05	CRAN (R 4.3.0)
pbivnorm	0.6.0	Loaded	2015-01-23	CRAN (R 4.3.0)
pillar	1.9.0	Loaded	2023-03-22	CRAN (R 4.3.0)
pkgbuild	1.4.2	Loaded	2023-06-26	CRAN (R 4.3.0)
pkgconfig	2.0.3	Loaded	2019-09-22	CRAN (R 4.3.0)
pkgload	1.3.3	Loaded	2023-09-22	CRAN (R 4.3.1)
plyr	1.8.9	Loaded	2023-10-02	CRAN (R 4.3.1)
png	0.1-8	Loaded	2022-11-29	CRAN (R 4.3.0)
prettyunits	1.2.0	Loaded	2023-09-24	CRAN (R 4.3.1)
processx	3.8.2	Loaded	2023-06-30	CRAN (R 4.3.0)
profvis	0.3.8	Loaded	2023-05-02	CRAN (R 4.3.0)
progress	1.2.2	Loaded	2019-05-16	CRAN (R 4.3.0)
promises	1.2.1	Loaded	2023-08-10	CRAN (R 4.3.0)
proxy	0.4-27	Loaded	2022-06-09	CRAN (R 4.3.0)

Table 5.1: List of all R Packages used in this book. *(continued)*

Package	Version	Attached or Loaded	Date/Publication	Source
ps	1.7.5	Loaded	2023-04-18	CRAN (R 4.3.0)
pscl	1.5.5.1	Loaded	2023-05-10	CRAN (R 4.3.0)
qgcomp	2.15.2	Loaded	2023-08-10	CRAN (R 4.3.0)
quadprog	1.5-8	Loaded	2019-11-20	CRAN (R 4.3.0)
quantreg	5.97	Loaded	2023-08-19	CRAN (R 4.3.0)
qvalue	2.34.0	Loaded	2023-10-26	Bioconductor
r.jive	2.4	Loaded	2020-11-17	CRAN (R 4.3.0)
R6	2.5.1	Loaded	2021-08-19	CRAN (R 4.3.0)
RColorBrewer	1.1-3	Loaded	2022-04-03	CRAN (R 4.3.0)
Rcpp	1.0.11	Loaded	2023-07-06	CRAN (R 4.3.0)
RcppParallel	5.1.7	Loaded	2023-02-27	CRAN (R 4.3.0)
remotes	2.4.2.1	Loaded	2023-07-18	CRAN (R 4.3.0)
reshape2	1.4.4	Loaded	2020-04-09	CRAN (R 4.3.0)
rjson	0.2.21	Loaded	2022-01-09	CRAN (R 4.3.0)
rlang	1.1.1	Loaded	2023-04-28	CRAN (R 4.3.0)
rmarkdown	2.25	Loaded	2023-09-18	CRAN (R 4.3.1)
RMediation	1.2.2	Loaded	2023-05-12	CRAN (R 4.3.0)
rprojroot	2.0.3	Loaded	2022-04-02	CRAN (R 4.3.0)
rstudioapi	0.15.0	Loaded	2023-07-07	CRAN (R 4.3.0)
S4Vectors	0.38.2	Loaded	2023-09-24	Bioconductor
scales	1.2.1	Loaded	2022-08-20	CRAN (R 4.3.0)
scalreg	1.0.1	Loaded	2019-01-25	CRAN (R 4.3.0)
selectiveInference	1.2.5	Loaded	2019-09-07	CRAN (R 4.3.0)
sessioninfo	1.2.2	Loaded	2021-12-06	CRAN (R 4.3.0)
shape	1.4.6	Loaded	2021-05-19	CRAN (R 4.3.0)
shiny	1.7.5.1	Loaded	2023-10-14	CRAN (R 4.3.1)
SparseM	1.81	Loaded	2021-02-18	CRAN (R 4.3.0)
stringi	1.7.12	Loaded	2023-01-11	CRAN (R 4.3.0)
survival	3.5-7	Loaded	2023-08-14	CRAN (R 4.3.0)
tidyselect	1.2.0	Loaded	2022-10-10	CRAN (R 4.3.0)
timechange	0.2.0	Loaded	2023-01-11	CRAN (R 4.3.0)
tzdb	0.4.0	Loaded	2023-05-12	CRAN (R 4.3.0)
urlchecker	1.0.1	Loaded	2021-11-30	CRAN (R 4.3.0)
utf8	1.2.4	Loaded	2023-10-22	CRAN (R 4.3.1)
vctrs	0.6.4	Loaded	2023-10-12	CRAN (R 4.3.1)
viridisLite	0.4.2	Loaded	2023-05-02	CRAN (R 4.3.0)
withr	2.5.2	Loaded	2023-10-30	CRAN (R 4.3.1)
xfun	0.40	Loaded	2023-08-09	CRAN (R 4.3.0)
xtable	1.8-4	Loaded	2019-04-21	CRAN (R 4.3.0)
xtune	2.0.0	Loaded	2023-06-18	CRAN (R 4.3.0)
yaml	2.3.7	Loaded	2023-01-23	CRAN (R 4.3.0)

5.3 *Software Environment*

This book was developed on the following platform.

Setting	Value
version	R version 4.3.1 (2023-06-16)
os	macOS Sonoma 14.1
system	aarch64, darwin20
ui	X11
language	(EN)
collate	en_US.UTF-8
ctype	en_US.UTF-8
tz	America/Los_Angeles
date	2023-11-02
pandoc	3.1.1 @ /Applications/RStudio.app/Contents/Resources/app/quarto/bin/tools/ (via rmarkdown)

6 Supplemental Code

6.1 Code for plotting Integrated/Quasi Mediation Analysis Results

The following code is provided as an example of how to plot the results of the integrated/quasi-mediation analysis. Since the results of these models depend critically upon the structure of the data used for the models, these functions may require alterations to obtain informative figures when analyzing other datasets.

6.1.1 Early Integration of omics datasets

```r
#' Plot Sankey Diagram for LUCID in Early integration
#'
#' Given an object of class from LUCID
#'
#' @param lucid_fit1  an object of class from LUCID
#' @param text_size  size of the text in sankey diagram
#'
#' @return a Sankey Diagram for LUCID in Early integration
#'
#' @import dplyr
#' @importFrom ggplot2 ggplot



sankey_early_integration <- function(lucid_fit1, text_size = 15) {
  # Get sankey dataframe ----
  get_sankey_df <- function(x,
                            G_color = "dimgray",
                            X_color = "#eb8c30",
                            Z_color = "#2fa4da",
                            Y_color = "#afa58e",
                            pos_link_color = "#67928b",
                            neg_link_color = "#d1e5eb",
                            fontsize = 10) {
    K <- x$K
    var.names <- x$var.names
    dimG <- length(var.names$Gnames)
```

```r
dimZ <- length(var.names$Znames)
valueGtoX <- as.vector(t(x$res_Beta[, -1]))
valueXtoZ <- as.vector(t(x$res_Mu))
valueXtoY <- as.vector(x$res_Gamma$beta)[1:K]

# GtoX
GtoX <- data.frame(
  source = rep(x$var.names$Gnames, K),
  target = paste0("Latent Cluster",
                 as.vector(sapply(1:K, function(x) rep(x, dimG)))),
  value = abs(valueGtoX),
  group = as.factor(valueGtoX > 0))

# XtoZ
XtoZ <- data.frame(
  source = paste0("Latent Cluster",
                 as.vector(sapply(1:K,
                                  function(x) rep(x, dimZ)))),
  target = rep(var.names$Znames,
               K), value = abs(valueXtoZ),
  group = as.factor(valueXtoZ >
                     0))

# subset top 25% of each omics layer
top25<- XtoZ %>%
  filter(source == "Latent Cluster1") %>%
  mutate(omics = case_when(grepl("cg", target) ~ "Methylation",
                           grepl("tc", target) ~ "Transcriptome",
                           grepl("miR", target) ~ "miRNA")) %>%
  group_by(omics) %>%
  arrange(desc(value)) %>%
  slice(1:7) %>%
  ungroup()

XtoZ_sub<- XtoZ %>%
  filter(target %in% top25$target)


# XtoY
XtoY <- data.frame(source = paste0("Latent Cluster", 1:K),
                   target = rep(var.names$Ynames, K), value = abs(valueXtoY),
                   group = as.factor(valueXtoY > 0))
links <- rbind(GtoX, XtoZ_sub, XtoY)
# links <- rbind(GtoX, XtoZ, XtoY)
```

```
  nodes <- data.frame(
    name = unique(c(as.character(links$source),
                    as.character(links$target))),
    group = as.factor(c(rep("exposure",
                            dimG), rep("lc", K), rep("biomarker", nrow(XtoZ_sub)/2), "outcome
  # group = as.factor(c(rep("exposure",
  # dimG), rep("lc", K), rep("biomarker", dimZ), "outcome")))
  ## the following two lines were used to exclude covars from the plot
  links <- links %>% filter(!grepl("cohort", source) &
                              !grepl("age", source) &
                              !grepl("fish", source) &
                              !grepl("sex", source))
  nodes <- nodes %>% filter(!grepl("cohort", name) &
                              !grepl("age", name) &
                              !grepl("fish", name) &
                              !grepl("sex", name))

  links$IDsource <- match(links$source, nodes$name) - 1
  links$IDtarget <- match(links$target, nodes$name) - 1

  color_scale <- data.frame(
    domain = c("exposure", "lc", "biomarker",
               "outcome", "TRUE", "FALSE"),
    range = c(G_color, X_color,
              Z_color, Y_color, pos_link_color, neg_link_color))

  sankey_df = list(links = links,
                   nodes = nodes)
  return(sankey_df)
}
# 1. Get sankey dataframes ----
sankey_dat <- get_sankey_df(lucid_fit1)
n_omics <- length(lucid_fit1$var.names$Znames)
# link data
links <- sankey_dat[["links"]]
# node data
nodes <- sankey_dat[["nodes"]]

nodes1 <- nodes %>%
  mutate(group = case_when(str_detect(name,"Cluster") ~ "lc",
                           str_detect(name, "cg") ~ "CpG",
                           str_detect(name, "outcome") ~ "outcome",
                           str_detect(name, "pro") ~ "Prot",
                           str_detect(name, "met") ~ "Met",
                           str_detect(name, "tc") ~ "TC",
                              str_detect(name, "miR") ~ "miRNA",
                              str_detect(name, "G1") ~ "exposure"),
         name = ifelse(name == "G1", "Hg",name))
links1 <- links %>%
  mutate(source = ifelse(source == "G1", "Hg",source))
# 6. Plotly Version ----

## 6.1 Set Node Color Scheme: ----
color_pal_sankey <- matrix(
  c("exposure", sankey_colors$range[sankey_colors$domain == "exposure"],
    "lc",        "#b3d8ff",
    "CpG",      sankey_colors$range[sankey_colors$domain == "layer1"],
    "TC",       sankey_colors$range[sankey_colors$domain == "layer2"],
    "miRNA", sankey_colors$range[sankey_colors$domain == "layer3"],
    "outcome",  sankey_colors$range[sankey_colors$domain == "Outcome"]),
  ncol = 2, byrow = TRUE) %>%
  as_tibble(.name_repair = "unique") %>%
  janitor::clean_names() %>%
  dplyr::rename(group = x1, color = x2)

# Add color scheme to nodes
nodes_new_plotly <- nodes1 %>%
  left_join(color_pal_sankey) %>%
  mutate(
    x = case_when(
      group == "exposure" ~ 0,
      str_detect(name, "Cluster") ~ 1/3,
      str_detect(name, "cg")|
        str_detect(name, "tc")|
        str_detect(name, "miR")|
        str_detect(name, "outcome")~ 2/3
    ))

nodes_new_plotly1 <- nodes_new_plotly %>%
  # Modify names of features for plotting
 dplyr::select(group, color, x, name)%>%
  mutate(name = case_when(name == "value" ~ "<b>Hg</b>",
                          name == "Latent Cluster1" ~ "<b>Joint Omics\nProfile 0</b>",
                          name == "Latent Cluster2" ~ "<b>Joint Omics\nProfile 1</b>",
                          TRUE ~ name))


## 6.2 Get links for Plotly, set color ----
links_new <- links1  %>%
  mutate(
        link_color = case_when(
          # Ref link color
          value == 0 ~      "#f3f6f4",
          # # Cluster
          # str_detect(source, "Cluster1") &  group == TRUE  ~  "#706C6C",
          # str_detect(source, "Cluster1") &  group == FALSE ~  "#D3D3D3",
          # str_detect(source, "Cluster2") &  group == TRUE  ~  "#706C6C",
          # str_detect(source, "Cluster2") &  group == FALSE ~  "#D3D3D3",
          ##############
          # Exposure
          str_detect(source, "Hg") &  group == TRUE  ~  "red",
              # Outcome
          str_detect(target, "outcome") &  group == TRUE  ~  "red",
          # Methylation
          str_detect(target, "tc") &  group == TRUE  ~  "#bf9000",
          str_detect(target, "tc") &  group == FALSE ~  "#ffd966",
          # Transcriptome
          str_detect(target, "cg") &  group == TRUE  ~  "#38761d",
          str_detect(target, "cg") &  group == FALSE ~  "#b6d7a8",
          # proteome
          str_detect(target, "miR") &  group == TRUE  ~  "#a64d79",
          str_detect(target, "miR") &  group == FALSE ~  "#ead1dc",
          ##
          group == FALSE ~ "#D3D3D3", # Negative association
          group == TRUE ~  "#706C6C")) # Positive association

links_new1<- links_new %>%
 dplyr::select(colnames(links_new), target)

plotly_link <- list(
  source = links_new1$IDsource,
  target = links_new1$IDtarget,
  value = links_new1$value+.00000000000000000000001,
  color = links_new1$link_color)

# Get list of nodes for Plotly
plotly_node <- list(
  label = nodes_new_plotly1$name,
  color = nodes_new_plotly1$color,
  pad = 15,
  thickness = 20,
  line = list(color = "black",width = 0.5),
  x = nodes_new_plotly1$x,
  # y = c(0.01,
  #       0.3, 0.7, # clusters
```

```
    #       seq(from = .01, to = 1, by = 0.04)[1:(dimZ * 0.25)], # biomaker
    #       .95
    y = c(0.01,
          0.1, 0.5, # clusters
          seq(from = .05, to = 1, by = 0.04)[1:21],
          # seq(from = (.01+0.06*7), to = 1, by = 0.08)[1:5],
          # 0.9,
          # biomaker
          0.98
  ))


  ## 6.3 Plot Figure ----
  (fig <- plot_ly(
    type = "sankey",
    domain = list(
      x =  c(0,1),
      y =  c(0,1)),
    orientation = "h",
    node = plotly_node,
    link = plotly_link))

  (fig <- fig %>% layout(
    # title = "Basic Sankey Diagram",
    font = list(
      size = text_size
    ))
  )
  return(fig)
}
```

6.1.2 Intermediate Integration of omics datasets

```
#' Plot Sankey Diagram for LUCID in parallel
#' @param lucidus_fit  an object of class from LUCID
#' @param sankey_colors a matrix including colors for each item in related sankey diagram
#' @param text_size  size of the text in sankey diagram
#' @param n_z_ftrs_to_plot an vector with numbers of features to show
#' in the sankey diagram for each omic layer
#'
#' @return a Sankey Diagram for LUCID in parallel
#'
#' @import dplyr
#' @importFrom ggplot2 ggplot
```

```r
#' @importFrom purrr map
#' @importFrom tidyr as_tibble
#' @importFrom dplyr left_join
plot_lucid_in_parallel_plotly<- function(lucidus_fit,
                                         sankey_colors,
                                         text_size = 10,
                                         n_z_ftrs_to_plot = NULL){
  # Get number of clusters, layers, etc.
  K <- lucidus_fit$K
  dimG <- lucidus_fit$res_Beta$Beta[[1]] %>% ncol()-1
  n_layers   <- length(lucidus_fit$res_Beta$Beta)

  # Get top omics features based on effect size
  if(!is.null(n_z_ftrs_to_plot)){
    top_ftrs <- vector("list", n_layers)
    for(i in seq_along(top_ftrs)){
      top_ftrs[[i]] <- lucidus_fit$res_Mu[[i]] %>%
        as.data.frame() %>%
        rownames_to_column("name") %>%
        mutate(effect_size = abs(V1) + abs(V2)) %>%
        arrange(desc(effect_size))
      top_ftr_nms <- top_ftrs[[i]]$name[1:n_z_ftrs_to_plot[i]]
      lucidus_fit$res_Mu[[i]] <-
        lucidus_fit$res_Mu[[i]][
          rownames(lucidus_fit$res_Mu[[i]]) %in% top_ftr_nms, ]
    }
  }

  mu_lst <- purrr::map(lucidus_fit$res_Mu,
                       ~as.data.frame(.x) %>%
                        rownames_to_column("name"))
  names(mu_lst) <- paste0("layer", c(1:n_layers))
  dimZ <- purrr::map(mu_lst, ncol) %>% as.numeric()-1
  n_features <- purrr::map(mu_lst, nrow) %>% as.numeric()
  names(n_features) <- paste0("layer", c(1:n_layers))
  # Names of features and set order of omics features
  names_features <- bind_rows(mu_lst, .id = "color_group") %>%
    rowwise() %>%
    mutate(sum = sum(abs(V1)+abs(V2)),
           pos_c2 = if_else(V2>0, "pos", "neg")) %>%
    group_by(color_group, pos_c2) %>% arrange(-sum, .by_group = TRUE) %>% ungroup() %>%
    mutate(rnum = row_number()) %>%
    group_by(name) %>% slice_head() %>% ungroup() %>%
    arrange(color_group, rnum) %>%
    dplyr::select(name, color_group)
```

```r
# Values for g --> x association
valueGtoX <- c(lapply(lucidus_fit$res_Beta$Beta,
                      function(x)(x[-1])) %>%
                 unlist(),
               rep(0, dimG*n_layers))

# For Cluster 2 (which needs effect estimates):
valueGtoX_c1 <- do.call(rbind, lucidus_fit$res_Beta$Beta)[,-1] %>%
  as_tibble() %>%
  dplyr::mutate(layer = str_c("(Layer ", row_number(), ")"),
                cluster = "Cluster 2")

# For cluster 1 (ref. cluster, effect est = 0):
valueGtoX_c2 <- valueGtoX_c1 %>%
  mutate(across(where(is.numeric), ~0),
         cluster = "Cluster 1")

# combine, pivot longer, and create source and target columns
GtoX <- bind_rows(valueGtoX_c1, valueGtoX_c2) %>%
  mutate(target = str_c(cluster, layer, sep = " ")) %>%
  pivot_longer(cols = setdiff(colnames(valueGtoX_c1),
                              c("layer", "cluster")),
               names_to = "source", values_to = "value") %>%
  mutate(color_group = as.factor(value > 0),
         value = abs(value)) %>%
  dplyr::select(source, target, value, color_group) %>%
  as.data.frame()

valueXtoZ <- c(lapply(lucidus_fit$res_Mu,
                      function(x)x[, 1]) %>%
                 unlist(),
               lapply(lucidus_fit$res_Mu,
                      function(x)x[, 2]) %>%
                 unlist())

valueXtoY <- c(rep(0, n_layers),
               # rep(lucidus_fit$res_Delta$Delta$mu[1] / n_layers, n_layers),
               lucidus_fit$res_Gamma$Gamma$mu[-1])

# n features in each layer
XtoZ <- data.frame(source = c(rep("Cluster 1 (Layer 1)", n_features[1]),
                              rep("Cluster 1 (Layer 2)", n_features[2]),
                              rep("Cluster 1 (Layer 3)", n_features[3]),
                              # rep("Cluster 1 (Layer 4)", n_features[4]),
                              rep("Cluster 2 (Layer 1)", n_features[1]),
                                rep("Cluster 2 (Layer 2)", n_features[2]),
                                rep("Cluster 2 (Layer 3)", n_features[3])
                                # rep("Cluster 2 (Layer 4)", n_features[4])
),
target = rep(c(lapply(lucidus_fit$res_Mu,
                      rownames) %>% unlist()),
             K[1]),
value = abs(valueXtoZ),
color_group = as.factor(valueXtoZ > 0))

# To change the outcome from left to right hand side, flip source and target
XtoY <- data.frame(target = rep("Outcome", 2*n_layers),
                   source = c("Cluster 1 (Layer 1)",
                              "Cluster 1 (Layer 2)",
                              "Cluster 1 (Layer 3)",
                              # "Cluster 1 (Layer 4)",
                              "Cluster 2 (Layer 1)",
                              "Cluster 2 (Layer 2)",
                              "Cluster 2 (Layer 3)"
                              # "Cluster 2 (Layer 4)"
                   ),
                   value = abs(valueXtoY),
                   color_group = as.factor(valueXtoY > 0))

# create Sankey diagram
# Create Links ----
links <- rbind(GtoX, XtoZ, XtoY) %>%
  mutate(
    # Group: one of exposure, clusters, or outcomes
    # (doesn't include Z.order by desired order)
    source_group = case_when(
      str_detect(source, "Cluster") ~ "2_Cluster",
      # source == "Outcome" ~ "3_outcome", # removed when moving outcome to right
      TRUE ~ "1_exposure"),
    # Source Omics Layer: lc1-lc4 (for omics layers), outcome, or other
    source_layer = case_when(
      str_detect(source, "Layer 1") ~ "lc1",
      str_detect(source, "Layer 2") ~ "lc2",
      str_detect(source, "Layer 3") ~ "lc3",
      str_detect(source, "Layer 4") ~ "lc4",
      # source == "Outcome" ~ str_sub(target, start = -3, end = -2),  # removed when moving o
      TRUE ~ "exposure"),
    # Source group_ for color (one of: exposure, : lc1-lc4 (for omics layers), outcome, or ot
    color_group_node = if_else(source == "Outcome",
                               "Outcome",
                               source_layer)) %>%
  group_by(source_group) %>%
  arrange(source_layer, .by_group = TRUE) %>%
  ungroup() %>%
  dplyr::select(source, target, value, color_group, color_group_node)


# Create Nodes ----
nodes <- links %>%
  dplyr::select(source, color_group_node) %>%
  mutate(rownum = row_number()) %>%
  rename(name = source,
         color_group = color_group_node) %>%
  # Add outcome (only if outcome is on right side)
  bind_rows(data.frame(name = "Outcome", color_group = "Outcome")) %>%
  group_by(name) %>%
  slice_head() %>%
  ungroup() %>%
  arrange(rownum) %>%
  dplyr::select(-rownum) %>%
  # Add feature names
  bind_rows(names_features) %>%
  mutate(id = row_number()-1) %>%
  left_join(sankey_colors, by = c( "color_group"= "domain"))

# Join links and nodes for color names -----
links <- links %>%
  left_join(nodes %>%
              dplyr::select(id, name),
            by = c("source" = "name")) %>%
  rename(source_id = id) %>%
  dplyr::select(source_id, everything()) %>%
  left_join(nodes %>%
              dplyr::select(id, name),
            by = c("target" = "name")) %>%
  rename(target_id = id) %>%
  dplyr::select(source_id,source, target_id,everything())


# Manually change colors ----
links <- links  %>%
  mutate(
    link_color = case_when(
      # Ref link color
      value == 0 ~      "#f3f6f4",
```

```r
      # Outcome
      str_detect(target, "Outcome") &  color_group == TRUE  ~  "red",
      # Methylation
      str_detect(source, "Layer 2") &  color_group == TRUE  ~  "#bf9000",
      str_detect(source, "Layer 2") &  color_group == FALSE ~  "#ffd966",
      # Transcriptome
      str_detect(source, "Layer 1") &  color_group == TRUE  ~  "#38761d",
      str_detect(source, "Layer 1") &  color_group == FALSE ~  "#b6d7a8",
      # mirna
      str_detect(source, "Layer 3") &  color_group == TRUE  ~  "#a64d79",
      str_detect(source, "Layer 3") &  color_group == FALSE ~  "#ead1dc",

      links$color_group == FALSE ~ "#d9d2e9", # Negative association
      links$color_group == TRUE ~  "red"))

## change node names
nodes <- nodes %>%
  mutate(name = case_when(name == "value" ~ "<b>Hg</b>",
                          name == "Cluster 1 (Layer 1)" ~ "<b>Methylation\nProfile 0</b>",
                          name == "Cluster 2 (Layer 1)" ~ "<b>Methylation\nProfile 1</b>",
                          name == "Cluster 1 (Layer 2)" ~ "<b>Transcriptome\nProfile 0</b>",
                          name == "Cluster 2 (Layer 2)" ~ "<b>Transcriptome\nProfile 1</b>",
                          name == "Cluster 1 (Layer 3)" ~ "<b>miRNA\nProfile 0</b>",
                          name == "Cluster 2 (Layer 3)" ~ "<b>miRNA\nProfile 1</b>",
                          TRUE ~ name),
         x = case_when(
           name == "Hg" ~ 0,
           str_detect(name, "Methylation") |
             str_detect(name, "Transcript") |
             str_detect(name, "miRNA") ~ 1/3,
           str_detect(name, "cg")|
             str_detect(name, "tc")|
             str_detect(name, "miR")|
             str_detect(name, "Outcome") ~ 2/3))

(fig <- plot_ly(
  type = "sankey",
  orientation = "h",
  domain = list(
    x =  c(0,0.8),
    y =  c(0,1)),
  # arrangement = "snap",
  node = list(
    label = nodes$name,
    color = nodes$range,
      pad = 15,
      thickness = 20,
      line = list(
        color = "black",
        width = 0.5
      ),
      x = nodes$x
    ),

    link = list(
      source = links$source_id,
      target = links$target_id,
      value =  links$value+.0000000000000000000001,
      # label = links$source,
      color = links$link_color
    )
  )
  )

  fig <- fig %>% layout(
    font = list(
      size = text_size
    ),
    xaxis = list(showgrid = F, zeroline = F),
    yaxis = list(showgrid = F, zeroline = F)
  )

  fig
}



# Set Color Palettes
col_pal <- RColorBrewer::brewer.pal(n = 8, name = "Dark2")

# Set Sankey Colors ----
# Color pallet for sankey diagrams
sankey_colors <- matrix(c("exposure", col_pal[6],
                          "lc1",      col_pal[1],
                          "lc2",      col_pal[2],
                          "lc3",      col_pal[3],
                          "lc4",      col_pal[4],
                          "layer1",   col_pal[1],
                          "layer2",   col_pal[2],
                          "layer3",   col_pal[3],
```

```r
                          "layer4",   col_pal[4],
                          "Outcome",  col_pal[8],
                          "TRUE",     "#6372e0", # Blue
                          "FALSE",    "#d1d4ff", # Light grey
                          "pos_clus_to_out", "red",
                          "neg_clus_to_out", "#e4e5f2"),
                        byrow = TRUE, nrow = 14)

# Change to dataframe
colnames(sankey_colors) <- c("domain", "range")
sankey_colors <- as.data.frame(sankey_colors)
```

6.1.3 Late Integration of omics datasets

```r
#' Plot Sankey Diagram for LUCID in late integration

# Get sankey dataframe
get_sankey_df <- function(x,
                          G_color = "dimgray",
                          X_color = "#eb8c30",
                          Z_color = "#2fa4da",
                          Y_color = "#afa58e",
                          pos_link_color = "#67928b",
                          neg_link_color = "#d1e5eb",
                          fontsize = 7) {
  K <- x$K
  var.names <- x$var.names
  pars <- x$pars
  dimG <- length(var.names$Gnames)
  dimZ <- length(var.names$Znames)
  valueGtoX <- as.vector(t(x$res_Beta[, -1]))
  valueXtoZ <- as.vector(t(x$res_Mu))
  valueXtoY <- as.vector(x$res_Gamma$beta)[1:K]

  # GtoX
  GtoX <- data.frame(
    source = rep(x$var.names$Gnames, K),
    target = paste0("Latent Cluster",
                    as.vector(sapply(1:K, function(x) rep(x, dimG)))),
    value = abs(valueGtoX),
    group = as.factor(valueGtoX > 0))

  # XtoZ
  XtoZ <- data.frame(
  source = paste0("Latent Cluster",
                  as.vector(sapply(1:K,
                                   function(x) rep(x, dimZ)))),
  target = rep(var.names$Znames,
               K), value = abs(valueXtoZ),
  group = as.factor(valueXtoZ >
                      0))
# XtoY
XtoY <- data.frame(source = paste0("Latent Cluster", 1:K),
                   target = rep(var.names$Ynames, K), value = abs(valueXtoY),
                   group = as.factor(valueXtoY > 0))

links <- rbind(GtoX, XtoZ, XtoY)

nodes <- data.frame(
  name = unique(c(as.character(links$source),
                  as.character(links$target))),
  group = as.factor(c(rep("exposure",
                          dimG), rep("lc", K), rep("biomarker", dimZ), "outcome")))

## the following two lines were used to exclude covars from the plot
links <- links %>% filter(!grepl("cohort", source) &
                            !grepl("age", source) &
                            !grepl("fish", source) &
                            !grepl("sex", source))
nodes <- nodes %>% filter(!grepl("cohort", name) &
                            !grepl("age", name) &
                            !grepl("fish", name) &
                            !grepl("sex", name))


links$IDsource <- match(links$source, nodes$name) - 1
links$IDtarget <- match(links$target, nodes$name) - 1

color_scale <- data.frame(
  domain = c("exposure", "lc", "biomarker",
             "outcome", "TRUE", "FALSE"),
  range = c(G_color, X_color,
            Z_color, Y_color, pos_link_color, neg_link_color))

sankey_df = list(links = links,
                 nodes = nodes)

# p <- sankeyNetwork(
#   Links = sankey_df$links,
  #   Nodes = sankey_df$nodes,
  #   Source = "IDsource",
  #   Target = "IDtarget",
  #   Value = "value",
  #   NodeID = "name",
  #   colourScale = JS(sprintf("d3.scaleOrdinal()\n .domain(%s)\n .range(%s)\n ",
  #                            jsonlite::toJSON(color_scale$domain),
  #                            jsonlite::toJSON(color_scale$range))),
  #   LinkGroup = "group",
  #   NodeGroup = "group",
  #   sinksRight = FALSE,
  #   fontSize = fontsize)
  # p
  return(sankey_df)
}


# sankey_in_serial Function ----
sankey_in_serial <- function(lucid_fit1, lucid_fit2, lucid_fit3, color_pal_sankey, text_size =

  # 1. Get sankey dataframes ----
  sankey_dat1 <- get_sankey_df(lucid_fit1)
  sankey_dat2 <- get_sankey_df(lucid_fit2)
  sankey_dat3 <- get_sankey_df(lucid_fit3)

  n_omics_1 <- length(lucid_fit1$var.names$Znames)
  n_omics_2 <- length(lucid_fit2$var.names$Znames)
  n_omics_3 <- length(lucid_fit3$var.names$Znames)

  # combine link data
  lnks1_methylation <- sankey_dat1[["links"]] %>% mutate(analysis = "1_methylation")
  lnks2_miRNA  <- sankey_dat2[["links"]] %>% mutate(analysis = "2_miRNA")
  lnks3_transcription    <- sankey_dat3[["links"]] %>% mutate(analysis = "3_transcript")
  links <- bind_rows(lnks1_methylation, lnks2_miRNA, lnks3_transcription)

  # combine node data
  nodes1_methylation <- sankey_dat1[["nodes"]] %>% mutate(analysis = "1_methylation")
  nodes2_miRNA  <- sankey_dat2[["nodes"]] %>% mutate(analysis = "2_miRNA")
  nodes3_transcription    <- sankey_dat3[["nodes"]] %>% mutate(analysis = "3_transcript")
  nodes <- bind_rows(nodes1_methylation, nodes2_miRNA, nodes3_transcription)


  # 2. Modify analysis 1 ----
  # For analysis 1, latent clusters need to be renamed to names from analysis 2:
  ## 2.1 Get new and original latent cluster names (from the next analysis) ----
```

```r
names_clusters_1 <- data.frame(
  name_og = c("Latent Cluster1", "Latent Cluster2"),
  name_new = c("<b>Methylation\nProfile 0</b>", "<b>Methylation\nProfile 1</b>"))

## 2.2 Change link names ----
# Change link names and
lnks1_methylation_new <- sankey_dat1[["links"]] %>%
  mutate(
    analysis = "1_methylation",
    source = case_when(
      source == names_clusters_1$name_og[1] ~ names_clusters_1$name_new[1],
      source == names_clusters_1$name_og[2] ~ names_clusters_1$name_new[2],
      TRUE ~ source),
    target = case_when(
      target == names_clusters_1$name_og[1] ~ names_clusters_1$name_new[1],
      target == names_clusters_1$name_og[2] ~ names_clusters_1$name_new[2],
      TRUE ~ target)) %>%
  filter(target != "outcome")

## 2.3 Change node names ----
# first, change latent cluster names to analysis specific cluster names
nodes1_methylation_new <- sankey_dat1[["nodes"]] %>%
  mutate(
    name = case_when(
      name == names_clusters_1$name_og[1] ~ names_clusters_1$name_new[1],
      name == names_clusters_1$name_og[2] ~ names_clusters_1$name_new[2],
      TRUE ~ name),
    group = if_else(group == "biomarker", "CpG", as.character(group))) %>%
  filter(group != "outcome")


# Visualize
# sankeyNetwork(
#   Links = lnks1_methylation_new,
#   Nodes = nodes1_methylation_new,
#   Source = "IDsource", Target = "IDtarget",
#   Value = "value", NodeID = "name", LinkGroup = "group", NodeGroup = "group",
#   sinksRight = FALSE)


# 3. Modify analysis 2 ----
# For analysis 2, latent clusters need to be renamed to names from analysis 3:
## 3.1 Get new and og latent cluster names ----
names_clusters_2 <- data.frame(
  name_og = c("Latent Cluster1", "Latent Cluster2"),
  name_new = c("<b>miRNA\nProfile 0</b>", "<b>miRNA\nProfile 1</b>"))

## 3.2 Change cluster names ----
lnks2_miRNA_new <- sankey_dat2[["links"]] %>%
  mutate(
    analysis = "2_miRNA",
    source = case_when(
      source == names_clusters_2$name_og[1] ~ names_clusters_2$name_new[1],
      source == names_clusters_2$name_og[2] ~ names_clusters_2$name_new[2],
      TRUE ~ source),
    target = case_when(
      target == names_clusters_2$name_og[1] ~ names_clusters_2$name_new[1],
      target == names_clusters_2$name_og[2] ~ names_clusters_2$name_new[2],
      TRUE ~ target)) %>%
  filter(target != "outcome")

## 3.3 Change node names ----
nodes2_miRNA_new <- sankey_dat2[["nodes"]] %>%
  mutate(
    name = case_when(
      name == names_clusters_2$name_og[1] ~ names_clusters_2$name_new[1],
      name == names_clusters_2$name_og[2] ~ names_clusters_2$name_new[2],
      TRUE ~ name),
    group = case_when(group == "exposure" ~ "lc",
                      group == "biomarker" ~ "miRNA",
                      TRUE ~ as.character(group))) %>%
  filter(name != "outcome")

# Visualize
# sankeyNetwork(
#   Links = lnks2_transcript_new,
#   Nodes = nodes2_transcript_new,
#   Source = "IDsource", Target = "IDtarget",
#   Value = "value", NodeID = "name",
#   LinkGroup = "group", NodeGroup = "group",
#   sinksRight = FALSE)
##

# 4. Modify analysis 3 ----
# For analysis 2, latent clusters need to be renamed to names from analysis 3:
## 4.1 Get new and og latent cluster names ----
names_clusters_3 <- tibble(
  name_og = c("Latent Cluster1", "Latent Cluster2"),
  name_new = c("<b>Transcriptome\nProfile 0</b>", "<b>Transcriptome\nProfile 1</b>"))
```

```r
## 4.2 Change cluster names ----
lnks3_transcript_new <- sankey_dat3[["links"]] %>%
  mutate(
    analysis = "3_transcript",
    source = case_when(
      source == names_clusters_3$name_og[1] ~ names_clusters_3$name_new[1],
      source == names_clusters_3$name_og[2] ~ names_clusters_3$name_new[2],
      TRUE ~ source),
    target = case_when(
      target == names_clusters_3$name_og[1] ~ names_clusters_3$name_new[1],
      target == names_clusters_3$name_og[2] ~ names_clusters_3$name_new[2],
      TRUE ~ target))

## 4.3 Change node names ----
nodes3_transcript_new <- sankey_dat3[["nodes"]] %>%
  mutate(
    name = case_when(
      name == names_clusters_3$name_og[1] ~ names_clusters_3$name_new[1],
      name == names_clusters_3$name_og[2] ~ names_clusters_3$name_new[2],
      TRUE ~ name),
    group = case_when(group == "exposure" ~ "lc",
                      group == "biomarker" ~ "TC",
                      TRUE ~ as.character(group)))

# Test/Visualize
# sankeyNetwork(
#   Links = lnks3_protein_new,
#   Nodes = nodes3_protein_new,
#   Source = "IDsource", Target = "IDtarget",
#   Value = "value", NodeID = "name", LinkGroup = "group", NodeGroup = "group",
#   sinksRight = FALSE)



# 5. Combine analysis 1-3 ----

## 5.1 Final Links ----
links_all_1 <- bind_rows(lnks1_methylation_new,
                         lnks2_miRNA_new,
                         lnks3_transcript_new) %>%
  dplyr::select(-IDsource, -IDtarget)


### 5.1.1 Arrange by magnitude ----
omics_priority <- links_all_1 %>%
  filter(str_detect(source, "Profile 0"),
         str_detect(target, "Profile 0", negate = TRUE),
         str_detect(target, "Profile 1", negate = TRUE),
         str_detect(target, "outcome", negate = TRUE)) %>%
  group_by(source) %>%
  arrange(desc(group), desc(value), .by_group = TRUE) %>%
  mutate(omics_order = row_number()) %>%
  ungroup() %>%
  dplyr::select(target, omics_order)



links_all <- links_all_1 %>%
  left_join(omics_priority) %>%
  mutate(
    # arrange_me = if_else(is.na(omics_order),
    #                          "dont_arrange",
    #                          "arrange"),
    row_num = row_number(),
    # row_num_order_comb = if_else(is.na(omics_order),
    #                               row_num,
    #                               omics_order),
    row_num_to_add = if_else(is.na(omics_order),
                             as.numeric(row_num),
                             NA_real_) %>%
      zoo::na.locf(),
    order = if_else(is.na(omics_order),
                    row_num_to_add,
                    row_num_to_add+omics_order)
  ) %>%
  arrange(order)


### 5.1.2 Get new source and target IDs ----
# First, combine all layers, get unique identifier
node_ids <- tibble(name = unique(c(unique(links_all$source),
                                   unique(links_all$target)))) %>%
  mutate(ID = row_number()-1)

# Then combine with original data
links_new <- links_all %>%
  left_join(node_ids, by = c("source" = "name")) %>%
  dplyr::rename(IDsource = ID) %>%
  left_join(node_ids, by = c("target" = "name")) %>%
  dplyr::rename(IDtarget = ID)
```

```r
## 5.2 Final Nodes ----
nodes_new <- node_ids %>%
  dplyr::select(name) %>%
  left_join(bind_rows(nodes1_methylation_new,
                      nodes2_miRNA_new,
                      nodes3_transcript_new))
# remove duplicates
nodes_new_nodup <- nodes_new[!base::duplicated(nodes_new),] %>%
  base::as.data.frame()


# 6. Plotly Version ----
library(plotly)

# Add color scheme to nodes
nodes_new_plotly <- nodes_new_nodup %>%
  left_join(color_pal_sankey) %>%
  mutate(
    x = case_when(
      group == "exposure" ~ 0,
      str_detect(name, "Methylation") ~ 1/5,
      str_detect(name, "miRNA") |
        str_detect(group, "CpG") ~ 2/5,
      str_detect(name, "Transcriptome") |
        str_detect(group, "miRNA") ~ 3/5,
      str_detect(group, "TC") ~  4/5,
      str_detect(group, "outcome") ~ 4.5/5,
    ))


## 6.2 Get links for Plotly, set color ----
links_new <- links_new  %>%
  mutate(
    link_color = case_when(
      # Ref link color
      value == 0 ~     "#f3f6f4",
      # Methylation
      str_detect(target, "outcome") &  group == TRUE  ~  "red",

      str_detect(source, "Transcriptome") &  group == TRUE  ~  "#bf9000",
      str_detect(source, "Transcriptome") &  group == FALSE ~  "#ffd966",
      # Transcriptome
      str_detect(source, "Methylation") &  group == TRUE  ~  "#38761d",
      str_detect(source, "Methylation") &  group == FALSE ~  "#b6d7a8",
      # proteome
      str_detect(source, "miRNA") &  group == TRUE  ~  "#a64d79",
      str_detect(source, "miRNA") &  group == FALSE ~  "#ead1dc",

      links_new$group == FALSE ~ "#d9d2e9", # Negative association
      links_new$group == TRUE ~  "red")) # Positive association

plotly_link <- list(
  source = links_new$IDsource,
  target = links_new$IDtarget,
  value = links_new$value+.000000000000000000000001,
  color = links_new$link_color)


# Get list of nodes for Plotly
plotly_node <- list(
  label = nodes_new_plotly$name,
  color = nodes_new_plotly$color,
  pad = 15,
  thickness = 20,
  line = list(color = "black",width = 0.5),
  x = nodes_new_plotly$x,
  y = c(0.01,
        0.1, 0.3, # Methylation clusters
        .45, .55, # Transcriptome clusters
        .80, .95, # Proteome clusters
        seq(from = .01, to = 1, by = 0.035)[1:n_omics_1], # Cpgs (10 total)
        seq(from = 0.35, to = 1, by = 0.025)[1:n_omics_2], # miRNA (8 total)
        seq(from = 0.75, to = 1, by = 0.03)[1:n_omics_3], # Transcript (10 total)
        .95
  ))


## 6.3 Plot Figure ----
fig <- plot_ly(
  type = "sankey",
  domain = list(
    x =  c(0,1),
    y =  c(0,1)),
  orientation = "h",
  node = plotly_node,
  link = plotly_link)

(fig <- fig %>% layout(
  # title = "Basic Sankey Diagram",
    font = list(
      size = text_size
    ))
  )

  return(fig)
}
```

6.1.4 Plot Omics Profile

```
#' Plot of Omics profiles for each cluster using LUCID
#'
#' Given an object of class from LUCID
#'
#' @param fit an object of class from LUCID
#' @param integration_type type of integration, "Early" or "Intermediate"
#'
#' @return a figure of Omics profiles for each cluster using LUCID
#'
#' @import dplyr
#' @importFrom ggplot2 ggplot
#'
plot_omics_profiles <- function(fit, integration_type, omics_lst_data) {

  # Combines omics data into one dataframe
  omics_lst_df <- purrr::map(omics_lst_data, ~tibble::as_tibble(.x, rownames = "name"))

  # Get metadata file
  meta_df <- imap_dfr(omics_lst_df,
                      ~tibble(omic_layer = .y, ftr_name = names(.x))) |>
    filter(ftr_name != "name") |>
    mutate(omic_num = case_when(str_detect(omic_layer, "meth") ~ 1,
                                str_detect(omic_layer, "transc") ~ 2,
                                str_detect(omic_layer, "miR") ~ 3,
                                str_detect(omic_layer, "pro") ~ 4,
                                str_detect(omic_layer, "met") ~ 5))

  if(integration_type == "Early"){
    M_mean = as.data.frame(fit$res_Mu)
    M_mean$cluster = as.factor(1:2)
    # Reshape the data
    M_mean_melt <- M_mean %>%
      pivot_longer(cols = -cluster, names_to = "variable", values_to = "value")
  M_mean_melt <- M_mean_melt %>%
    mutate(cluster = paste0("Cluster ", cluster))
  # add color label for omics layer
  M_mean_melt = M_mean_melt %>%
    mutate(color_label = case_when(str_detect(variable,  "cg") ~ "1",
                                   str_detect(variable, "tc") ~ "2",
                                   TRUE ~ "3"))

  fig <- ggplot(M_mean_melt,
                aes(fill = color_label, y = value, x = variable)) +
    geom_bar(position="dodge", stat="identity") +
    ggtitle("Omics profiles for the two latent clusters") +
    facet_grid(rows = vars(cluster), scales = "free_y") +
    theme(legend.position="none") +
    geom_hline(yintercept = 0) +
    xlab("") +
    theme(text = element_text(size=10),
          axis.text.x = element_text(angle = 90, vjust = 1,
                                     hjust = 1),
          plot.margin = margin(10, 10, 10, 80),
          panel.background = element_rect(fill="white"),
          strip.background = element_rect(fill = "white"),
          axis.line.x = element_line(color = "black"),
          axis.line.y = element_line(color = "black"),) +
    scale_fill_manual(values = c("#2fa4da", "#A77E69", "#e7b6c1"))
} else if(integration_type == "Intermediate"){
  M_mean = as_tibble(fit$res_Mu[[1]], rownames = "variable") %>%
    bind_rows(as_tibble(fit$res_Mu[[2]], rownames = "variable")) %>%
    bind_rows(as_tibble(fit$res_Mu[[3]], rownames = "variable"))

  # Reorder results because mirna order is reversed
  M_mean1 <- M_mean %>%
    left_join(meta_df, by = c("variable" = "ftr_name")) %>%
    mutate(`Low Risk`  =  if_else(omic_layer == "miRna", V2, V1),
           `High Risk` =  if_else(omic_layer == "miRna", V1, V2)) %>%
    dplyr::select(-c("V1", "V2"))

  # Pivot longer for figure
  M_mean_l <- M_mean1 %>%
    pivot_longer(cols = c(`Low Risk`, `High Risk`),
                 names_to = "cluster",
                 values_to = "value")

  # add color label for omics layer
  M_mean2 = M_mean_l %>%
      mutate(color_label = case_when(omic_layer == "methylome" ~ "1",
                                     omic_layer == "transcriptome" ~ "2",
                                     omic_layer == "miRna" ~ "3"),
             low_high = if_else(str_detect(cluster, "Low"), 0,1),
             omic = if_else(omic_layer == "miRna",
                            "miR",
                            str_sub(omic_layer, end = 1) %>% toupper()),
             omic_cluster = str_c(omic, low_high))

    # Filter only the top ## differential expressed features
    M_mean2_top <- M_mean2 %>%
      group_by(variable) %>%
      filter(abs(value) == max(abs(value))) %>%
      ungroup() %>%
      arrange(max(abs(value))) %>%
      group_by(omic_layer) %>%
      slice_head(n=12) %>%
      ungroup()

    # Plots top 12 features
    fig <- ggplot(M_mean2  %>% filter(variable %in% M_mean2_top$variable),
                   aes(fill = color_label, y = value, x = variable)) +
      geom_bar(position="dodge", stat="identity") +
      ggtitle("Omics profiles for 2 latent clusters - Lucid in Parallel") +
      facet_grid(rows = vars(cluster),
                 cols = vars(omic_layer), scales = "free_x", space = "free") +
      theme(legend.position="none") +
      geom_hline(yintercept = 0) +
      xlab("") +
      theme(text = element_text(size=10),
            axis.text.x = element_text(angle = 90, vjust = 1,
                                       hjust = 1),
            plot.margin = margin(10, 10, 10, 80),
            panel.background = element_rect(fill="white"),
            strip.background = element_rect(fill = "white"),
            axis.line.x = element_line(color = "black"),
            axis.line.y = element_line(color = "black"),) +
      scale_fill_manual(values = c("#2fa4da", "#A77E69", "#e7b6c1"))
  }

  return(fig)
}
```

6.1.5 Change the Reference Cluster for Different LUCID Models

```r
#' reorder cluster estimated from LUCID
#'
#' @param model a model returned the function lucid
#' @param order the desired order of the cluster label. For example, if 2 clusters
#' are estiamted and you want to flip the cluster label, you should input c(2, 1).
#' The first element will be used as the reference cluster
#'
reorder_lucid <- function(model,
                          order) {
  # record parameters
  pars <- model$pars
  beta <- pars$beta
  mu <- pars$mu
  sigma <- pars$sigma
  gamma <- pars$gamma
  K <- model$K

  # 1 - reorder exposure effect
  # use the estimate from the
  ref_cluster <- order[1]
  beta <- t(t(beta) - beta[ref_cluster, ])[order, ]

  # 2 - reorder omic effect
  mu <- mu[order, ]
  var <- vector(mode = "list", length = K)
  for (i in 1:K) {
    var[[i]] <- sigma[[order[i]]]
  }

  # 3 - reorder outcome effect
  gamma$beta[1:K] <- gamma$beta[order]
  gamma$sigma <- gamma$sigma[order]

  model$pars <- list(beta = beta,
                     mu = mu,
                     sigma = sigma,
                     gamma = gamma)

  return(model)
}

# example ----
# G <- sim_data$G
# Z <- sim_data$Z
# Y_normal <- sim_data$Y_normal
# cov <- sim_data$Covariate
#
# # In this model, the exposure effect should be around 2
# fit <- lucid(G = G, Z = Z, Y = Y_normal, CoY = cov)
# summary_lucid(fit)
# flip the label of the 2 clusters,
# which is equivalent to use the second cluster as the reference cluster
# fit_reorder <- reorder_lucid(model = fit,
#                              order = c(2, 1))
# the exposure effect should be around 1/2


#' 2. change variable name in the sankey diagram ------------------
#' @title Visualize LUCID model through a Sankey diagram
#' @description In the Sankey diagram, each node either represents a variable (exposure,
#' omics or outcome) or a latent cluster. Each line represents an association. The
#' color of the node represents variable type, either exposure, omics or outcome.
#' The width of the line represents the effect size of a certain association; the
#' color of the line represents the direction of a certain association.
#'
#' @param x A LUCID model fitted by \code{\link{est_lucid}}
#' @param G_color Color of node for exposure
#' @param X_color Color of node for latent cluster
#' @param Z_color Color of node for omics data
#' @param G_name variable name for latent exposure variables
#' @param Z_name variable name for omic features
#' @param pos_link_color Color of link corresponds to positive association
#' @param neg_link_color Color of link corresponds to negative association
#' @param fontsize Font size for annotation
#'
#' @return A DAG graph created by \code{\link{sankeyNetwork}}
#'

plot_lucid_without_outcome <- function(x,
                       G_color = "dimgray",
                       X_color = "#eb8c30",
                       Z_color = "#2fa4da",
                       G_name = NULL,
                       Z_name = NULL,
                       pos_link_color = "#67928b",
                       neg_link_color = "#d1e5eb",
                       fontsize = 7
) {
```

```r
K <- x$K
var.names <- x$var.names
pars <- x$pars
dimG <- length(var.names$Gnames)
dimZ <- length(var.names$Znames)
valueGtoX <- as.vector(t(x$pars$beta[, -1]))
valueXtoZ <- as.vector(t(x$pars$mu))
# valueXtoY <- as.vector(x$pars$gamma$beta)[1:K]
if(is.null(G_name)) {
  G_name = x$var.names$Gnames
}
GtoX <- data.frame(source = rep(G_name, K),
                   target = paste0("Latent Cluster", as.vector(sapply(1:K, function(x) rep(x,
                   value = abs(valueGtoX),
                   group = as.factor(valueGtoX > 0))
if(is.null(Z_name)) {
  Z_name = var.names$Znames
}
XtoZ <- data.frame(source = paste0("Latent Cluster", as.vector(sapply(1:K, function(x) rep(x,
                   target = rep(Z_name, K),
                   value = abs(valueXtoZ),
                   group = as.factor(valueXtoZ > 0))
# if(is.null(Y_name)) {
#   Y_name = var.names$Ynames
# }
# XtoY <- data.frame(source = paste0("Latent Cluster", 1:K),
#                      target = rep(Y_name, K),
#                      value = abs(valueXtoY),
#                      group = as.factor(valueXtoY > 0))

links <- rbind(GtoX, XtoZ) #, XtoY
nodes <- data.frame(name = unique(c(as.character(links$source), as.character(links$target))),
                    group = as.factor(c(rep("exposure", dimG),
                                        rep("lc", K),
                                        rep("biomarker", dimZ))))
links$IDsource <- match(links$source, nodes$name)-1
links$IDtarget <- match(links$target, nodes$name)-1
color_scale <- data.frame(domain = c("exposure", "lc", "biomarker", "TRUE", "FALSE"),
                          range = c(G_color, X_color, Z_color, pos_link_color, neg_link_color

p <- sankeyNetwork(Links = links,
                   Nodes = nodes,
                   Source = "IDsource",
                   Target = "IDtarget",
                   Value = "value",
                     NodeID = "name",
                     colourScale = JS(
                       sprintf(
                         'd3.scaleOrdinal()
                        .domain(%s)
                        .range(%s)
                       ',
                       jsonlite::toJSON(color_scale$domain),
                       jsonlite::toJSON(color_scale$range)
                       )),
                     LinkGroup ="group",
                     NodeGroup ="group",
                     sinksRight = FALSE,
                     fontSize = fontsize)
  p
}

# example
# plot_lucid(fit)
# plot_lucid_without_outcome(fit)
# # you can also re-name exposure names and the omic feature name
# plot_lucid_without_outcome(fit,
#            G_name = paste0("PFAS_", 1:10))


#' reorder LUCID in Parallel model by specifying reference cluster ------------
# note: only works for K = 2 in each omic layer
# reference = c(1,1,2)
# lucidus_fit <- fit_reordered
reorder_lucid_parallel <- function(lucidus_fit,
                                   reference = NULL) {
  if(is.null(reference)) {
    warning("no reference specified, return the original model")
    return(lucidus_fit)
  }

  n_omic <- length(reference)

  # reorder beta
  GtoX <- lucidus_fit$res_Beta$Beta
  lucidus_fit$res_Beta$Beta <- lapply(1:n_omic, function(i) {
    (-1)^(reference[i] - 1) * GtoX[[i]] # if reference = 1, no changes;
    # if reference = 2, flip the reference and negate the estimates
  })
  # reorder mu
  XtoZ <- lucidus_fit$res_Mu
  lucidus_fit$res_Mu <- lapply(1:n_omic, function(i) {
    x <- c(1, 2) # order of clusters
    if(reference[i] == 2) {
      x <- c(2, 1)
      XtoZ[[i]][, x]
    } else{
      XtoZ[[i]][, x]
    }
  })
  # reorder gamma
  XtoY <- lucidus_fit$res_Gamma$Gamma$mu
  XtoY[1] <- XtoY[1] + sum(XtoY[-1] * (reference - 1)) # reference level using the new referenc
  XtoY[-1] <- (-1)^(reference - 1) * XtoY[-1] # if reference = 2, flip the estimates
  lucidus_fit$res_Gamma$Gamma$mu <- XtoY
  lucidus_fit$res_Gamma$fit$coefficients <- XtoY

  # return the object using the new reference
  return(lucidus_fit)
}
```

References

He, Jingxuan, and Chubing Zeng. 2023. "Xtune: Regularized Regression with Feature-Specific Penalties Integrating External Information." Computer Program. https://github.com/JingxuanH/xtune.

Lock, E. F., K. A. Hoadley, J. S. Marron, and A. B. Nobel. 2013. "JOINT AND INDIVIDUAL VARIATION EXPLAINED (JIVE) FOR INTEGRATED ANALYSIS OF MULTIPLE DATA TYPES." Journal Article. *Ann Appl Stat* 7 (1): 523–42. https://doi.org/10.1214/12-aoas597.

Peng, C., J. Wang, I. Asante, S. Louie, R. Jin, L. Chatzi, G. Casey, D. C. Thomas, and D. V. Conti. 2020. "A Latent Unknown Clustering Integrating Multi-Omics Data (LUCID) with Phenotypic Traits." Journal Article. *Bioinformatics* 36 (3): 842–50. https://doi.org/10.1093/bioinformatics/btz667.

Tofighi, D., and D. P. MacKinnon. 2011. "RMediation: An r Package for Mediation Analysis Confidence Intervals." Journal Article. *Behav Res Methods* 43 (3): 692–700. https://doi.org/10.3758/s13428-011-0076-x.

Vrijheid, Martine, Rémy Slama, Oliver Robinson, Leda Chatzi, Muireann Coen, Peter van den Hazel, Cathrine Thomsen, et al. 2014. "The Human Early-Life Exposome (HELIX): Project Rationale and Design." Journal Article. *Environmental Health Perspectives* 122 (6): 535–44. https://doi.org/10.1289/ehp.1307204.

Zeng, C., D. C. Thomas, and J. P. Lewinger. 2021. "Incorporating Prior Knowledge into Regularized Regression." Journal Article. *Bioinformatics* 37 (4): 514–21. https://doi.org/10.1093/bioinformatics/btaa776.

Zhang, H., Y. Zheng, Z. Zhang, T. Gao, B. Joyce, G. Yoon, W. Zhang, et al. 2016. "Estimating and Testing High-Dimensional Mediation Effects in Epigenetic Studies." Journal Article. *Bioinformatics* 32 (20): 3150–54. https://doi.org/10.1093/bioinformatics/btw351.