Supplemental Materials: Integrating Multi-Omics with Environmental data for Precision Health

Goodrich JA, Wang H, Jia Q, et. al

2023-08-20

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.

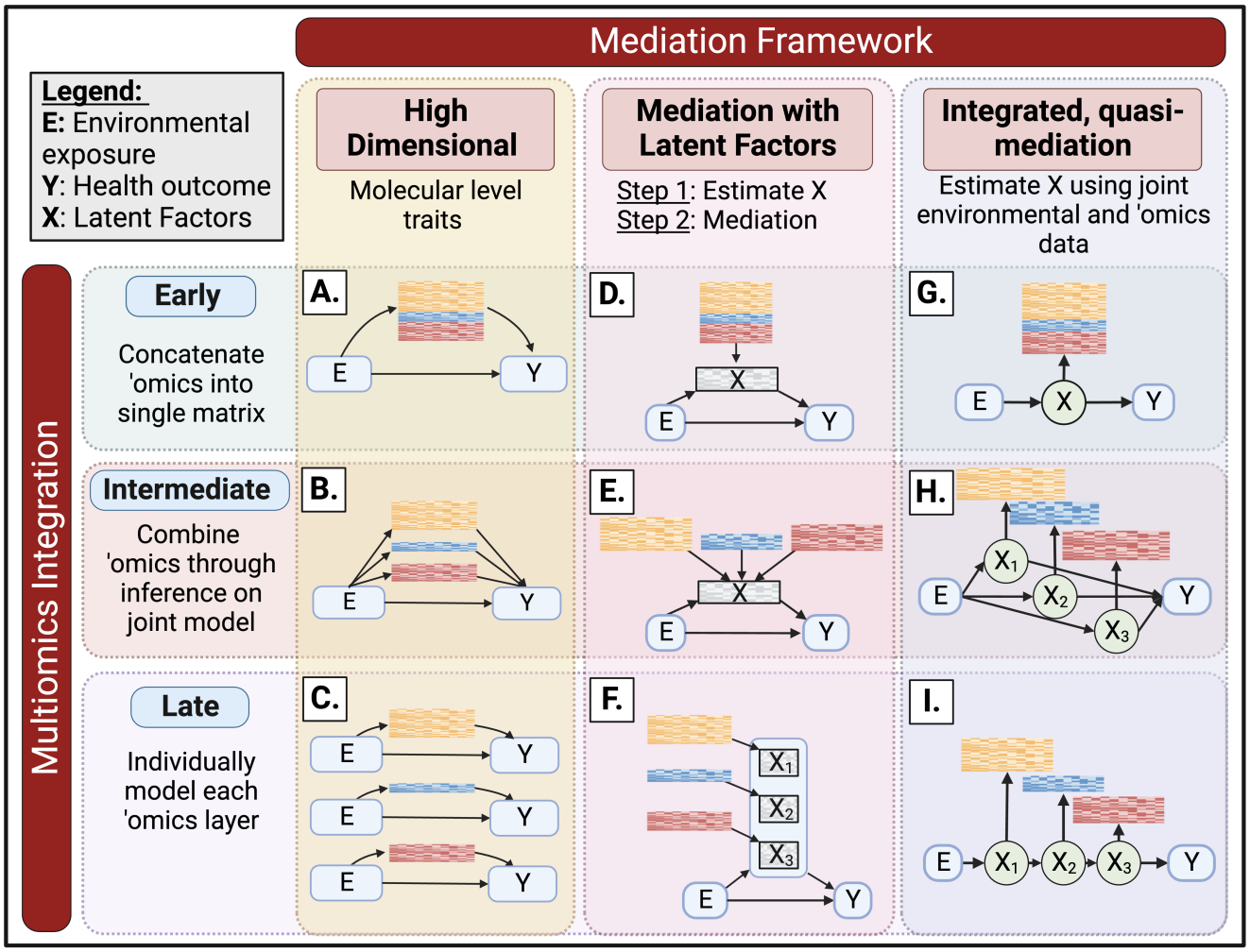


Figure 1: Conceptual diagram illustrating the analytic framework for mediation analysis with multiple omic layers.

1 Introduction

1.1 R Packages

Before starting, you will need the following 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 can be found in Chapter 5.

```
# General Packages:
library(tidyverse)
library(tools)
library(parallel)
library(boot)
library(table1)
# Packages for Plotting:
library(ggplot2)
library(cowplot)
library(ComplexHeatmap)
library(ggh4x)
# Packages for High Dimensional Mediation:
library(HIMA)
library(xtune)
library(RMediation)
library(glmnet)
# Packages for Mediation with Latent Factors:
library(r.jive)
# Packages for Quasi-mediation:
library(LUCIDus)
library(mclust)
library(networkD3)
library(plotly)
library(htmlwidgets)
library(glasso)
library(nnet)
library(progress)
library(jsonlite)
```

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

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

1.2 Custom Functions

The analyses in this book rely on several custom functions. The code for functions are provided in Chapter 6.

1.3 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).

You can read this data into R using the following code:

```
# Load simulated data
simulated_data <- read_rds(fs::path(dir_data_hg, "simulated_HELIX_data_2.RDS"))

# Define exposure and outcome name
covars <- c("e3_sex_None", "hs_child_age_yrs_None")

# Extract exposure and outcome data
# outcomes <- simulated_data[["phenotype"]]
exposure <- simulated_data[["phenotype"]]$hs_hg_m_scaled
outcome  <- simulated_data[["phenotype"]]$ck18_scaled

# Get numeric matrix of covariates
covs <- simulated_data[["phenotype"]][covars]
covs$e3_sex_None <- if_else(covs$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")
```

1.3.1 Descriptive Statistics

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

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%)

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

1.3.2 Mercury exposure and childhood MAFLD risk

```
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.

```
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)")
```

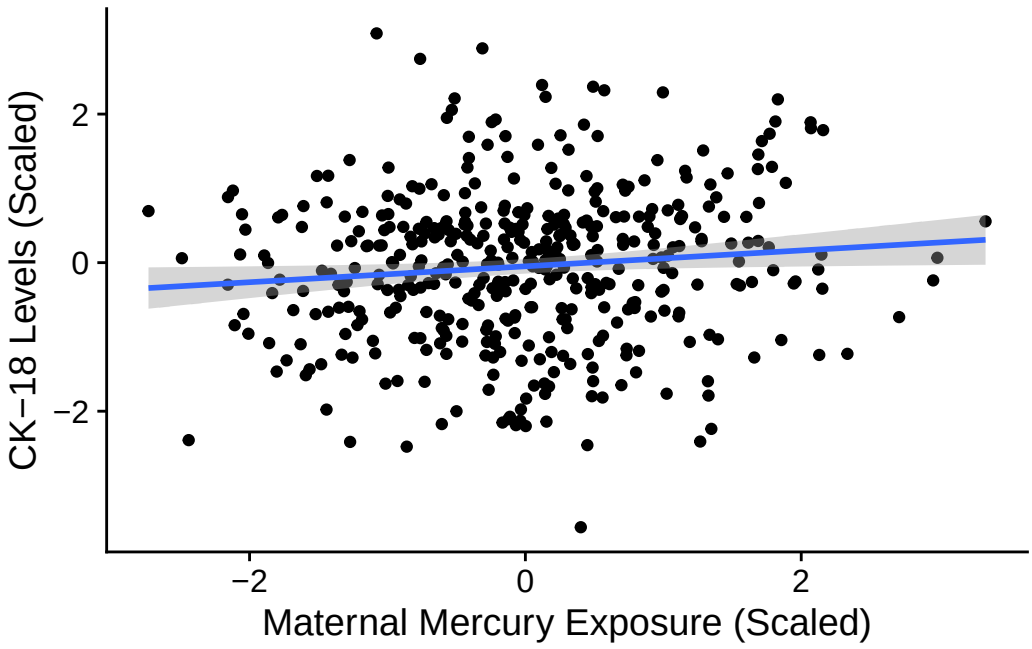


Figure 1.1: Association between maternal mercury and CK18 in the Simulated Data

1.3.3 Correlation of omics features

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

```
# 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 = toTitleCase(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"))
```

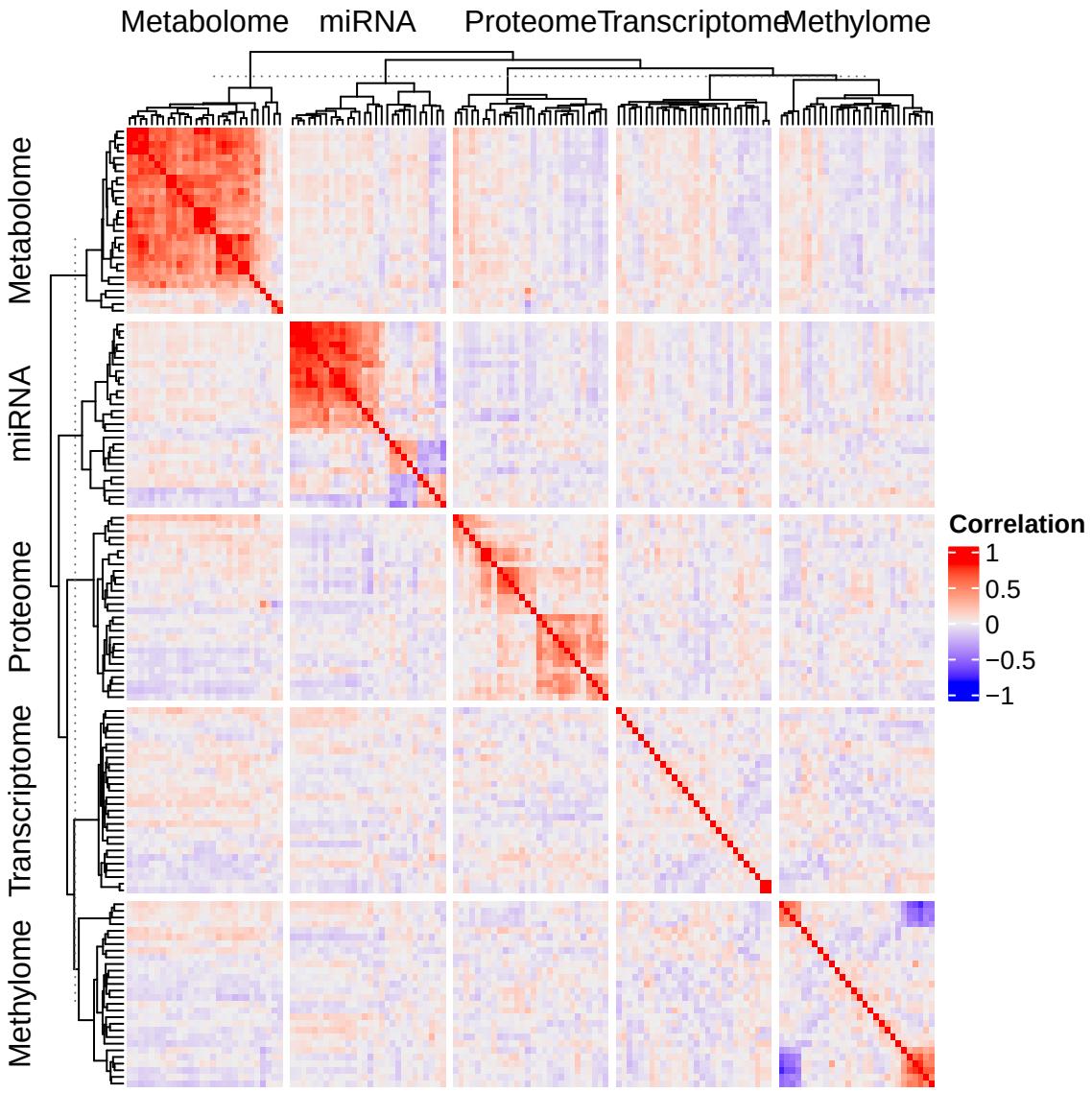


Figure 1.2: Heatmap illustrating the correlation of molecular features within and between different omics layers.

2 High Dimensional Mediation

2.1 *Early integration*

High dimensional 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). The full code for the `hima_early_integration` function is provided in Section 6.1.1 and `plot_hima` function is provided in Section 6.1.4.

```
# Run Analysis
result_hima_early <- hima_early_integration(exposure = exposure,
                                            outcome = outcome,
                                            omics_lst = omics_lst,
                                            covs = covs,
                                            Y.family = "gaussian",
                                            M.family = "gaussian")

# Plot Result
(plot_hima(result_hima_early))
```

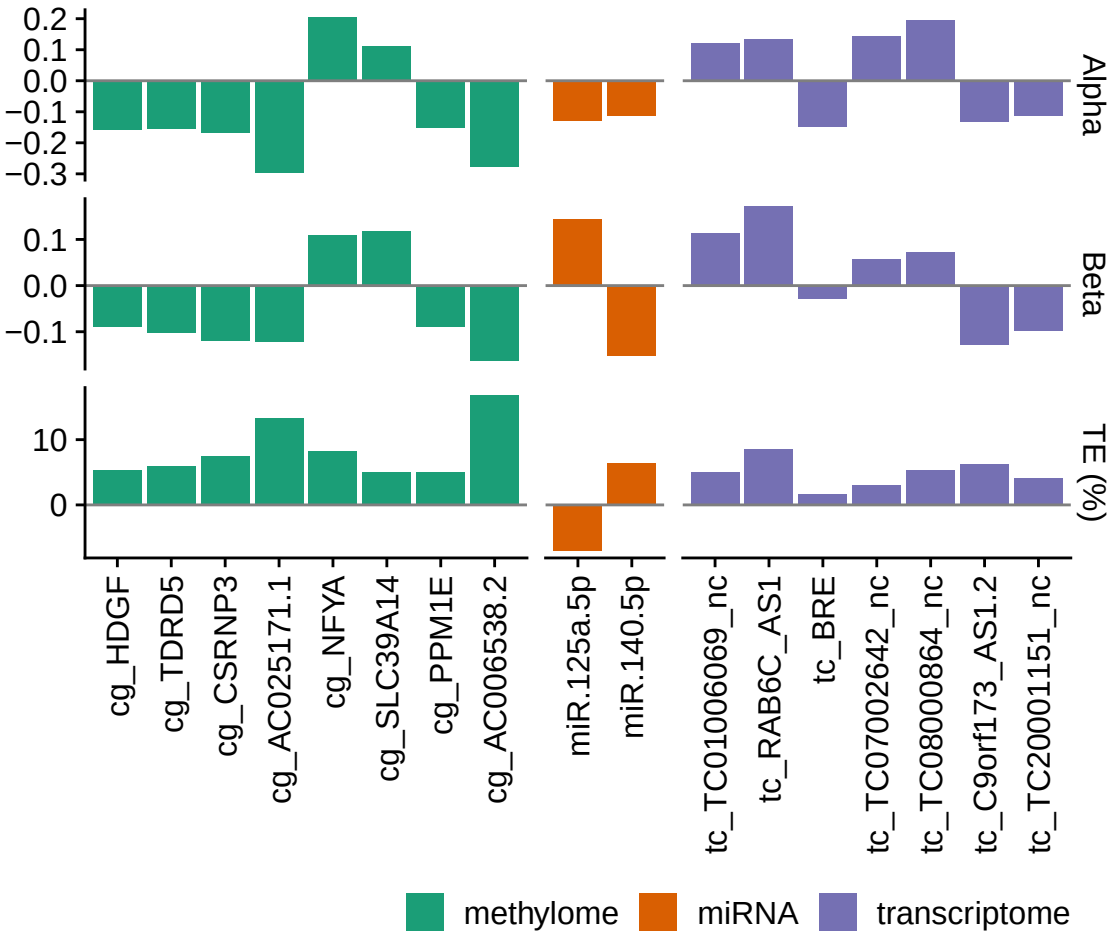


Figure 2.1: High dimensional mediation analysis with early integration and multiple omic layers identifies individual molecular features linking maternal mercury with childhood liver injury. Alpha represents the coefficient estimates of the exposure to the mediator, Beta indicates the coefficient estimates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

2.2 *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).

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

The full code for the `hima_intermediate_integration` function is provided in Section 6.1.2 and `plot_hima` function is provided in Section 6.1.4. *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 <- hima_intermediate_integration(
  omics_lst = omics_lst,
  covs = covs,
  outcome = outcome,
  exposure = exposure,
  n_boot = 12,
  Y.family = "gaussian")
```

```
# plot
(plot_hima(result_hima_intermediate))
```

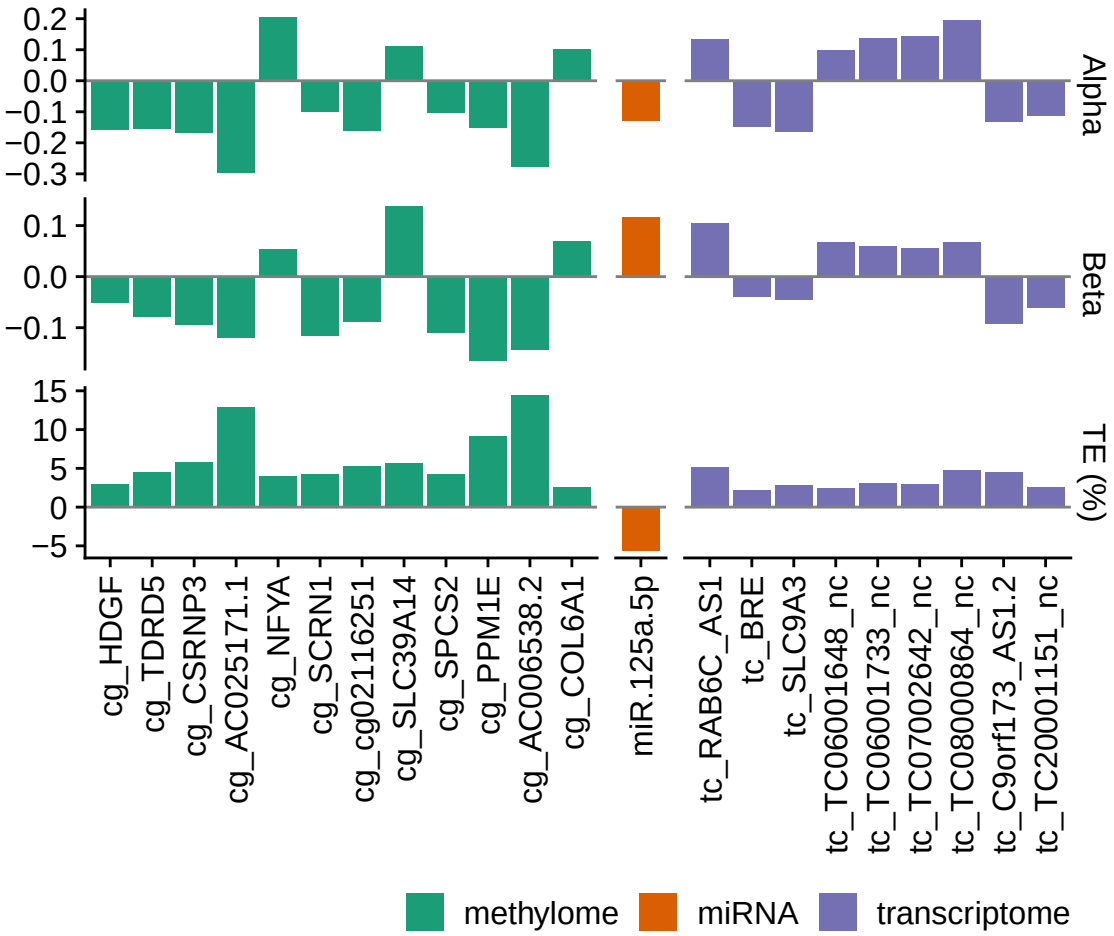


Figure 2.2: High dimensional mediation analysis with intermediate integration and multiple omic layers identifies individual molecular features linking maternal mercury with childhood liver injury. Alpha represents the coefficient estimates of the exposure to the mediator, Beta indicates the coefficient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

2.3 *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). The full code for the `hima_late_integration` function is provided in Section 6.1.3 and `plot_hima` function is provided in Section 6.1.4.

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

# plot
(plot_hima(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 TE (%) 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 Set up project for Mediation Analysis with Latent factors

```
source(fs::path(here::here(),"project_setup","directories.R"))
source(fs::path(dir_proj, "libraries.R"))
source(fs::path(dir_proj, "load_simu_data.R"))
source(fs::path(dir_proj, "functions","mediation_with_latent_fctrs_functions.R"))
source(fs::path(dir_proj, "functions","plot_mediation_lf_function.R"))

# options(knitr.table.format = "html")
```

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. The full code for the `med_lf_early` function is provided in Section 6.2.1 and `plot_med_lf` function is provided in Section 6.2.4.

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

```
# Run Analysis
result_med_with_latent_fctrs_early <-
  med_lf_early(exposure,
               outcome,
               omics_lst,
```

```
                covs = covs,
                Y.family = "gaussian",
                M.family = "gaussian",
                fdr.level = 0.05)
```

3.2.2 Plot Early Integration

```
# Plot
(plot_med_lf(result_med_with_latent_fctrs_early))
```

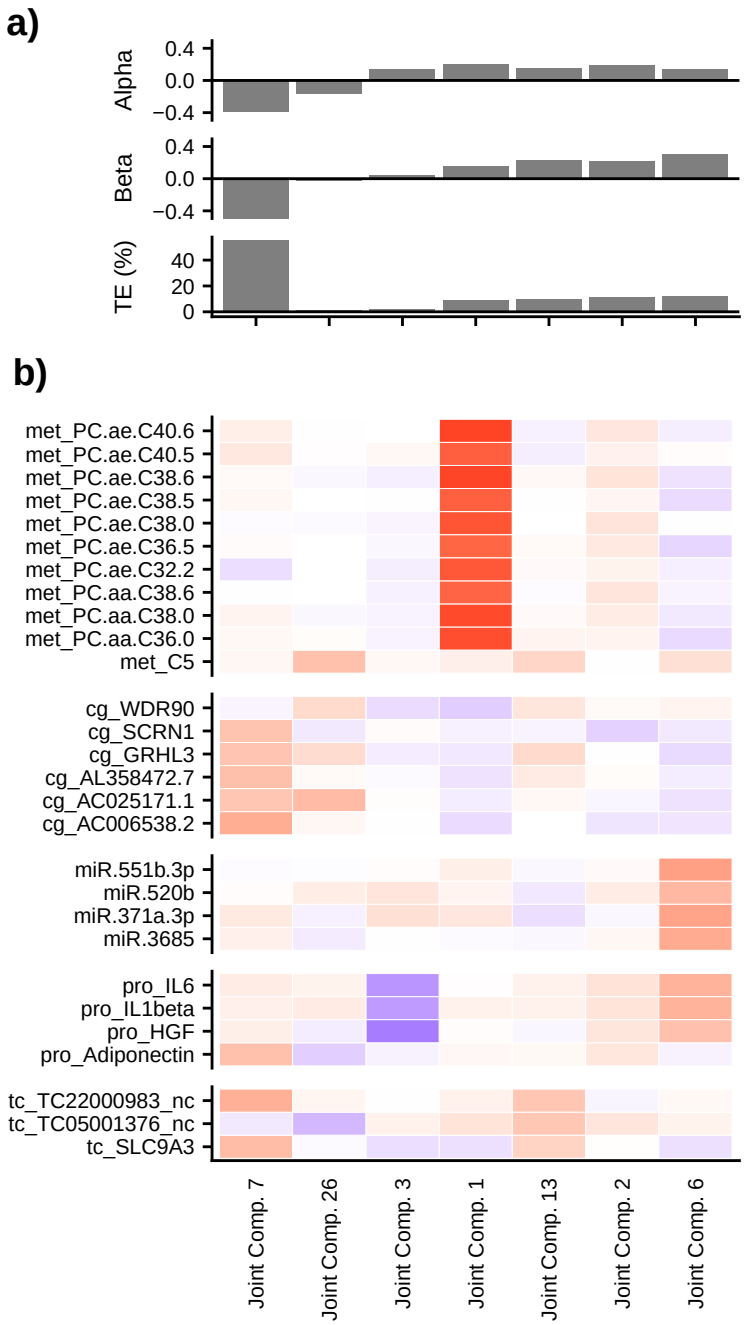


Figure 3.1: Mediation analysis with latent factors and early integration identifies joing components which mediate the association between maternal mercury and childhood liver injury. Panel A shows the mediation effects, where Alpha represents the coefficient estimates of the exposure to the mediator, Beta indicates the coefficient estimates of the mediators to the outcome, and TE (%) 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).The full code for the `med_lf_intermediate` function is provided in Section 6.2.2 and `plot_med_lf` function is provided in Section 6.2.4.

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

3.3.2 Plot Intermediate Integration

```
(plot_med_lf(result_med_with_latent_fctrs_JIVE))
```

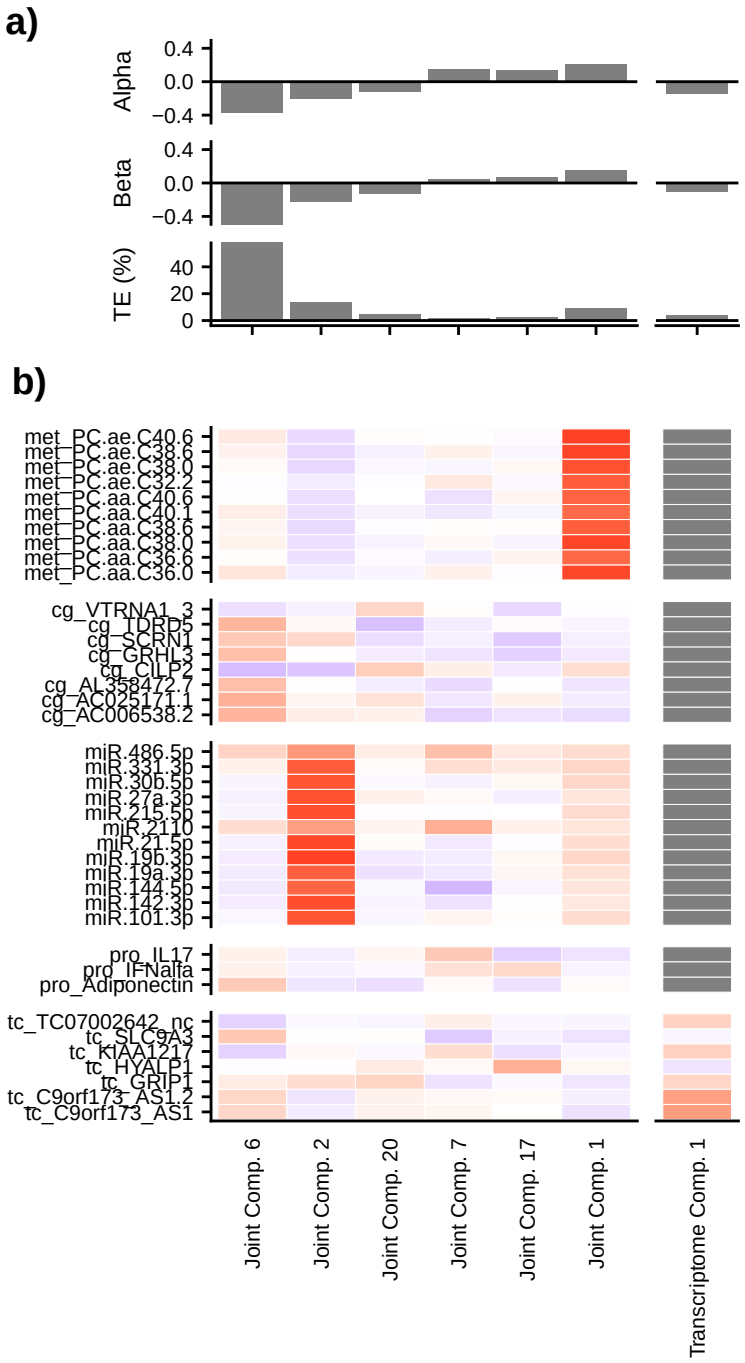


Figure 3.2: Mediation analysis with latent factors and intermediate integration identifies joint and individual variance componets which mediate the association between maternal mercury and childhood liver injury. Panel A shows the mediation effects, where Alpha represents the coefficient estimates of the exposure to the mediator, Beta indicates the coefficient estimates of the mediators to the outcome, and TE (%) 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. The full code for the `med_lf_late` function is provided in Section 6.2.3 and `plot_med_lf` function is provided in Section 6.2.4

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 <- med_lf_late(exposure,
                                                 outcome,
                                                 omics_lst,
                                                 covs = covs,
                                                 Y.family = "gaussian",
                                                 M.family = "gaussian",
                                                 fdr.level = 0.05)
```

3.4.2 Plot Late Integration

```
(plot_med_lf(result_med_with_latent_fctrs_late))
```

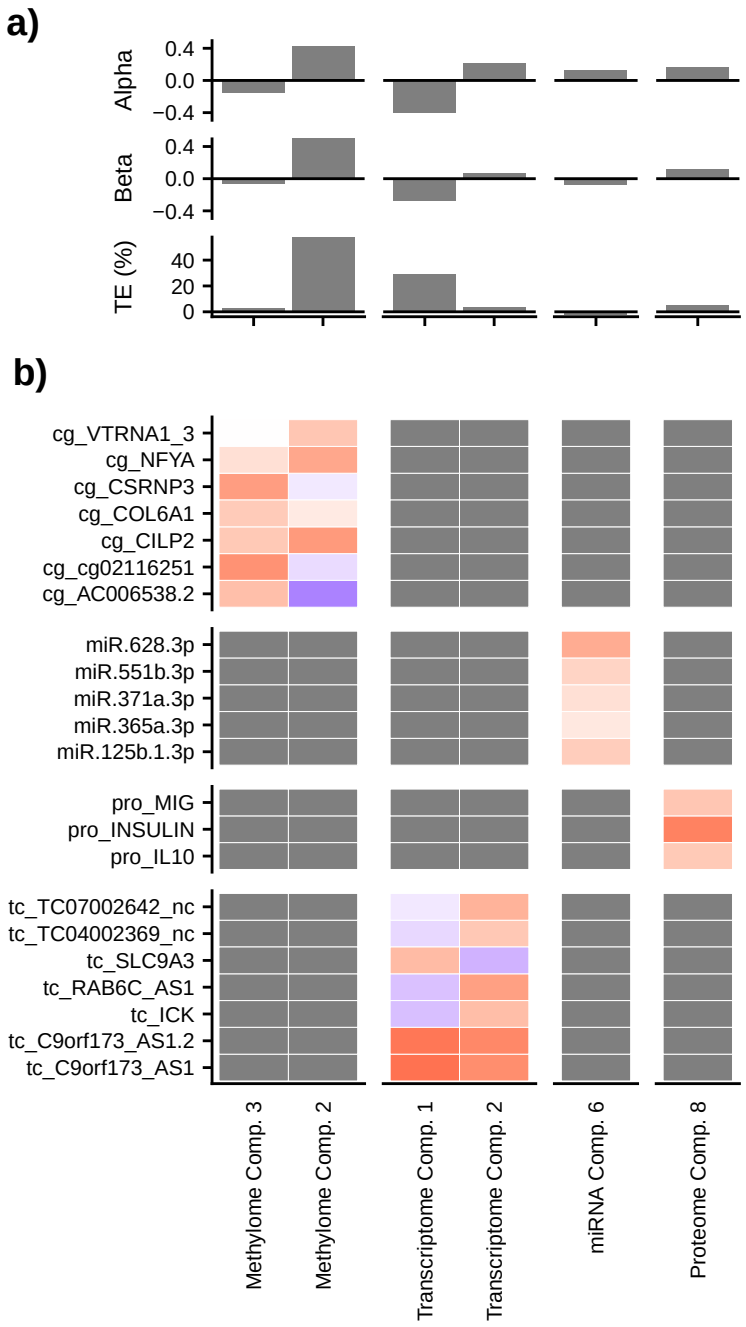


Figure 3.3: Mediation analysis with latent factors and late integration identifies features in each omics layer individually which mediates the association between maternal mercury and childhood liver injury. Panel A shows the mediation effects, where Alpha represents the coefficient estimates of the exposure to the mediator, Beta indicates the coefficient estimates of the mediators to the outcome, and TE (%) 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 *Early Integration*

In the Early Integration LUCID model, genomic/exposomic exposures G, other omics data Z and phenotype trait Y are integrated through a latent categorical variable X. 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.

4.1.1 Analysis: Early Integration

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

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 6.3.1

```
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")
## Initialize LUCID with random values from uniform distribution
## Fitting Early Integration LUCID model (K = 2, Rho_G = 0, Rho_Z_Mu = 0, Rho_Z_Cov = 0)
## .....Success: Early Integration LUCID converges!

# Sankey Diagram
p1 <- sankey_early_integration(fit1, text_size = 20)
```

4.1.2 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 6.3.4

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

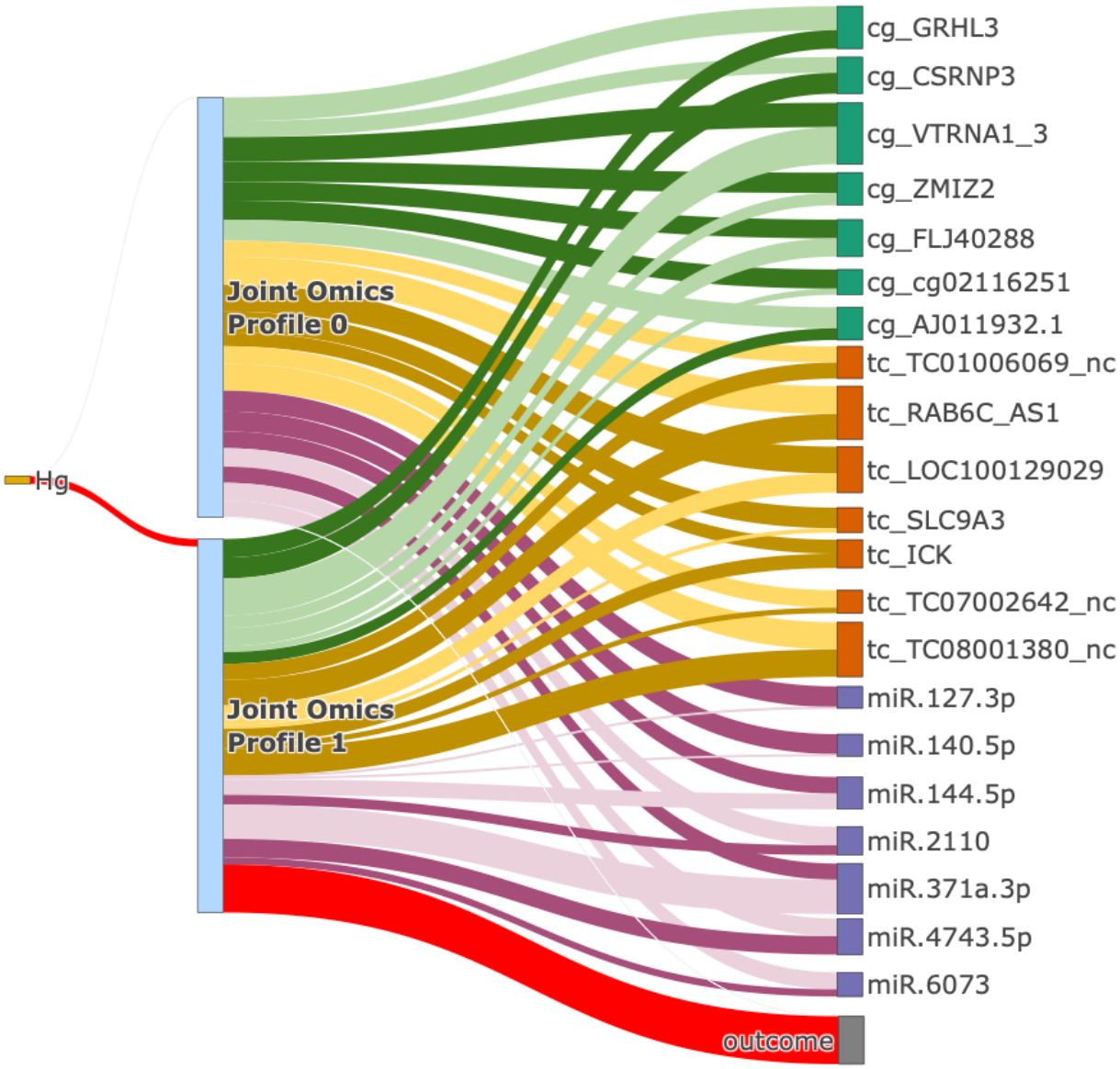
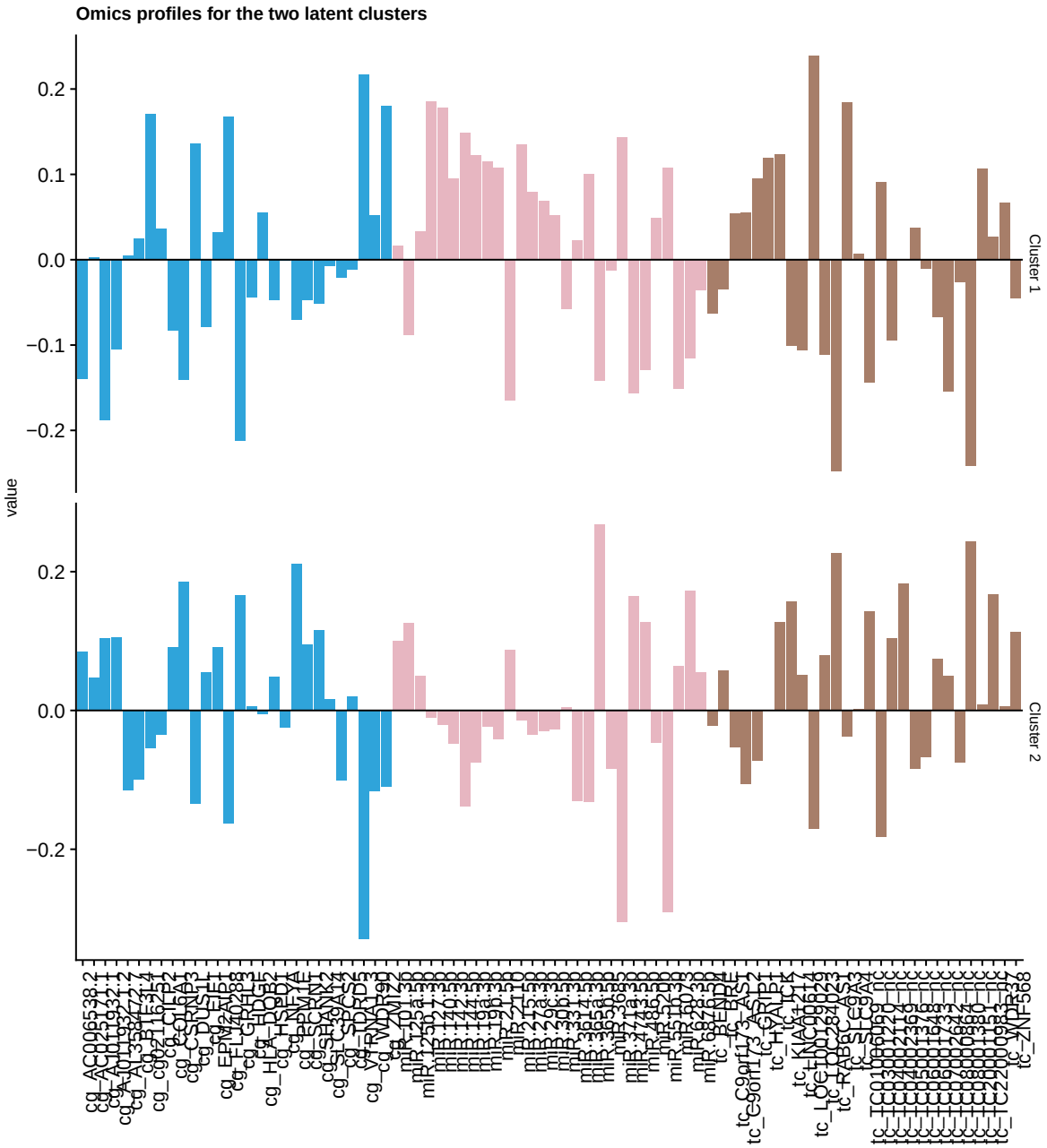


Figure 4.1: The Sankey Diagram for LUCID (Early Integration).



4.2 *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 $Z_1, \ldots, Z_a, \ldots, Z_m$ with corresponding dimensions $p_1, \ldots, p_a$,

. . . , 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,

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

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.

Set up the project for analysis.

Prepare data for quasi-mediation analysis with LUCID in Parallel.

```
# get feature meta data
omics_lst_df <- purrr::map(omics_lst, ~as_tibble(.x))

meta_df <- imap_dfr(omics_lst_df, ~tibble(omic_layer = .y, ftr_name = names(.x)))


omics_lst_nmd_ftr = omics_lst
# END NODE
```

4.2.1 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.

4.2.2 Sankey diagram

The full code for the `plot_lucid_in_parellel_plotly` function is provided in Section 6.3.2

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

4.2.3 Omics profiles for each cluster predicted by LUCID

The full code for the `plot_omics_profiles` function is provided in Section 6.3.4

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

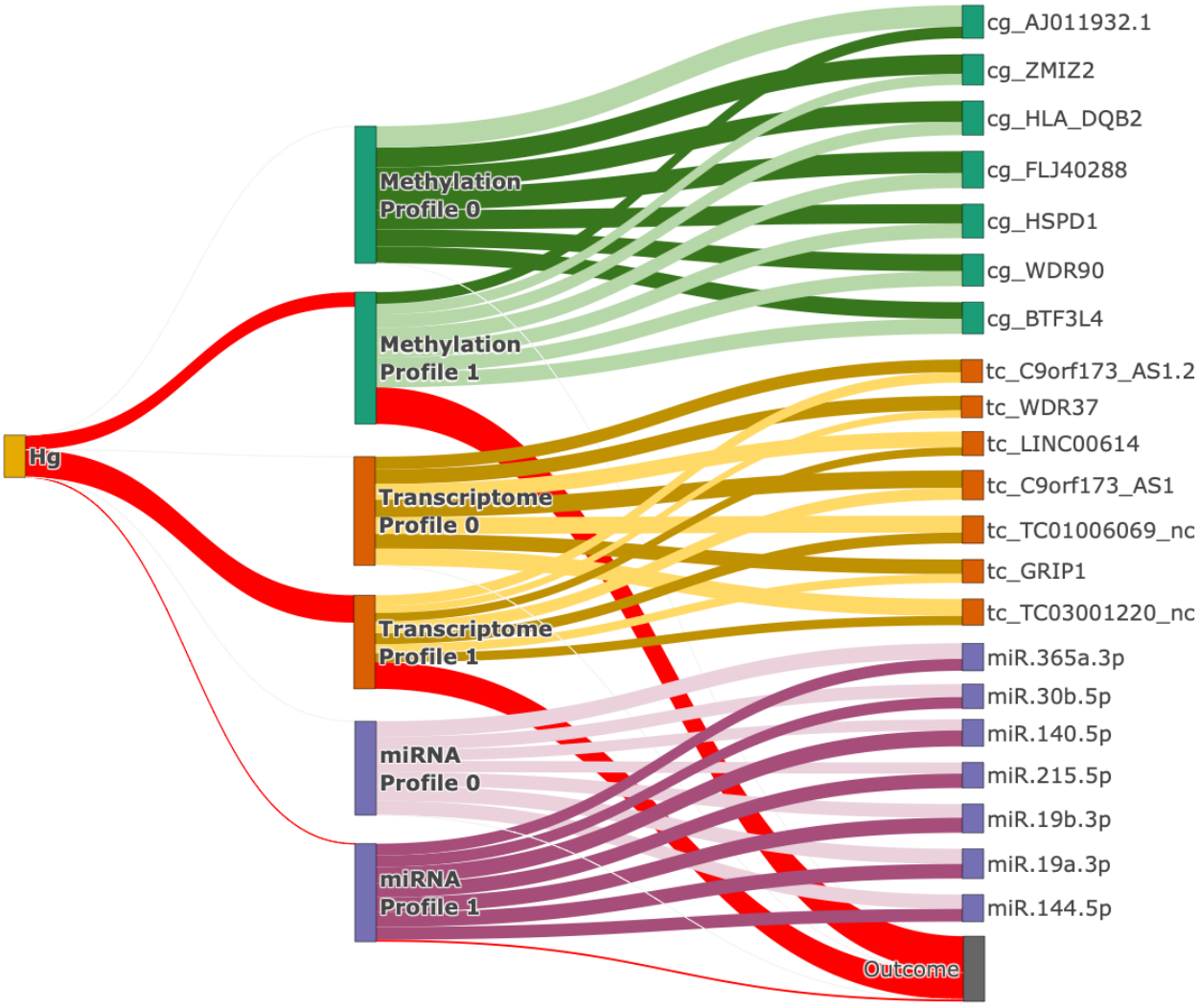
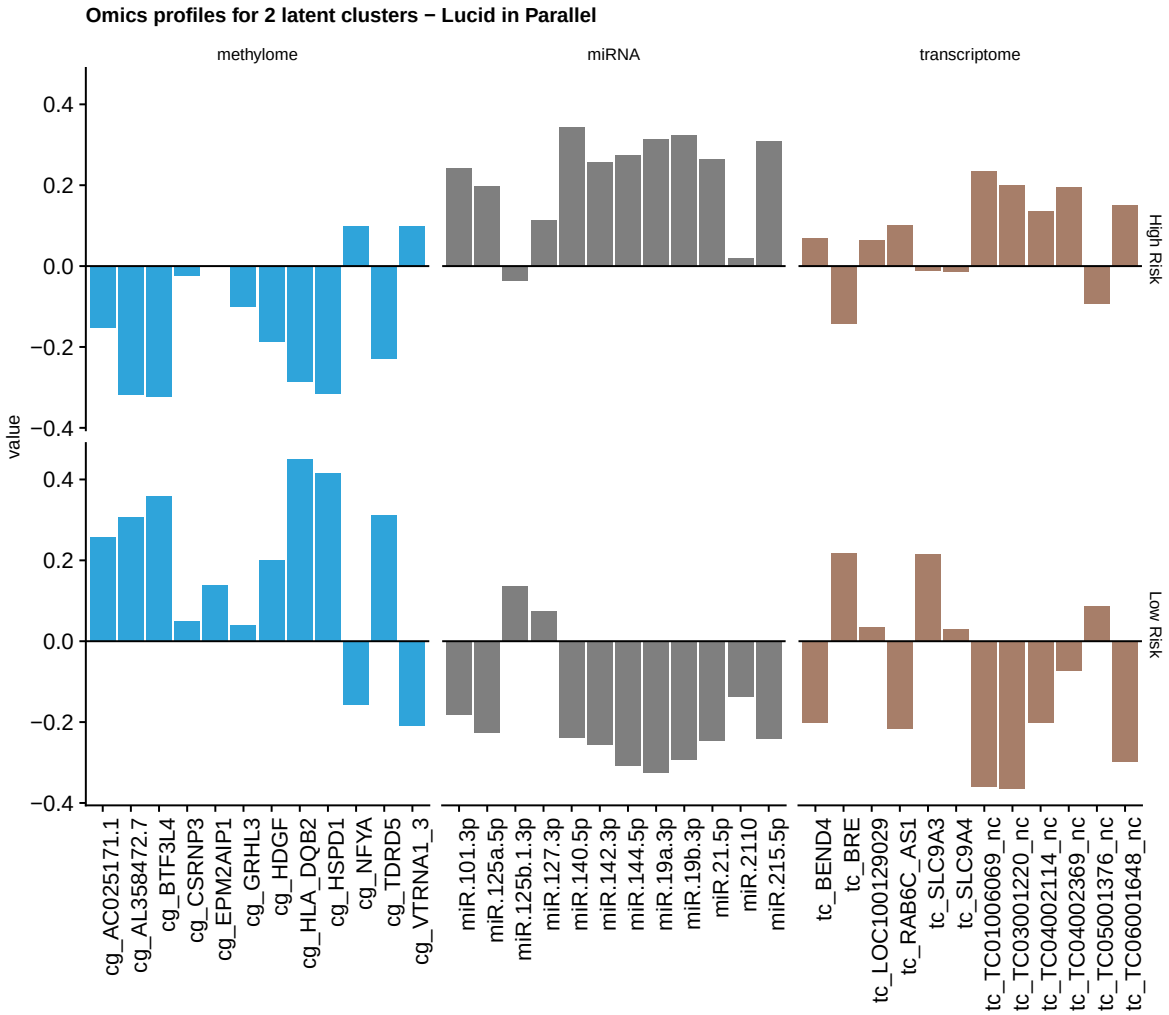


Figure 4.2: The Sankey Diagram for LUCID in Parallel (Intermediate Integration).



4.3 *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.

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.

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

#get the selection
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)
## Fitting LUCID in Serial model (Sub Model Number = 1)
## Initialize LUCID with mclust
## Fitting Early Integration LUCID model (K = 2, Rho_G = 0, Rho_Z_Mu = 10, Rho_Z_Cov = 0.3)
## ..................Success: Early Integration LUCID converges!
##
## Fitting LUCID in Serial model (Sub Model Number = 2)
## Initialize LUCID with mclust
## Fitting Early Integration LUCID model (K = 2, Rho_G = 0, Rho_Z_Mu = 10, Rho_Z_Cov = 0.3)
## ..........................................................................................
##
## Fitting LUCID in Serial model (Sub Model Number = 3)
## Initialize LUCID with mclust
## Fitting Early Integration LUCID model (K = 2, Rho_G = 0, Rho_Z_Mu = 10, Rho_Z_Cov = 0.3)
## ..............................................Success: Early Integration LUCID converges!
##
```

```
## Success: LUCID in Serial Model is constructed!

#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")
## Fitting LUCID in Serial model (Sub Model Number = 1)
## Initialize LUCID with random values from uniform distribution
## Fitting Early Integration LUCID model (K = 2, Rho_G = 0, Rho_Z_Mu = 0, Rho_Z_Cov = 0)
## ........................................Success: Early Integration LUCID converges!
##
## Fitting LUCID in Serial model (Sub Model Number = 2)
## Initialize LUCID with random values from uniform distribution
## Fitting Early Integration LUCID model (K = 2, Rho_G = 0, Rho_Z_Mu = 0, Rho_Z_Cov = 0)
## ..........................................................................................
##
## Fitting LUCID in Serial model (Sub Model Number = 3)
## Initialize LUCID with random values from uniform distribution
## Fitting Early Integration LUCID model (K = 2, Rho_G = 0, Rho_Z_Mu = 0, Rho_Z_Cov = 0)
## ..........................................................................................
##
## Success: LUCID in Serial Model is constructed!

# 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>"
```

4.3.1 Sankey Diagram

The full code for the `sankey_in_serial` function is provided in Section 6.3.3

```
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)
```

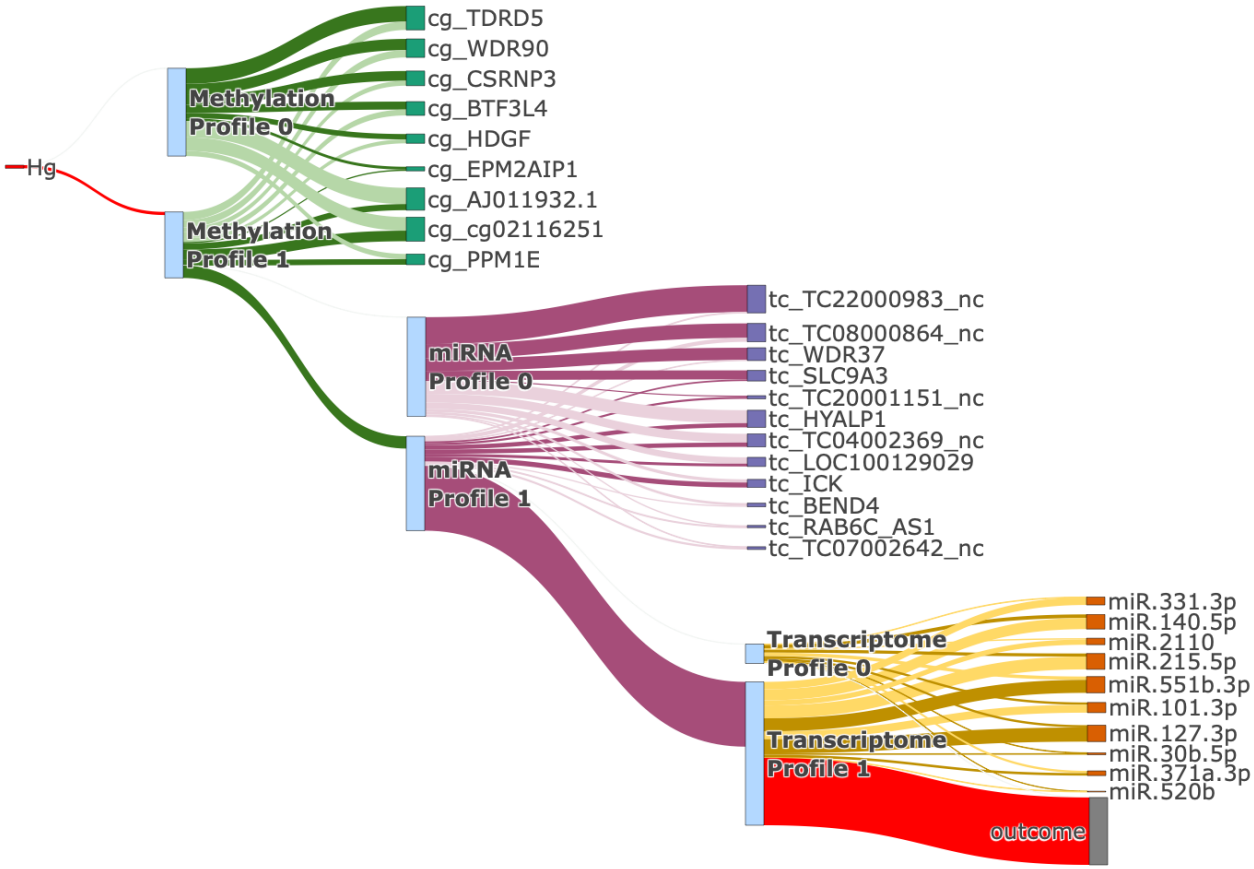


Figure 4.3: The Sankey Diagram for LUCID in Serial (Late Integration).

5 Software

5.1 *R packages*

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

Package	Version	Attached or Loaded	Date/Publication	Source
boot	1.3-28.1	Attached	2022-11-22	CRAN (R 4.3.1)
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)
e1071	1.7-13	Attached	2023-02-01	CRAN (R 4.3.0)
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)
glasso	1.11	Attached	2019-10-01	CRAN (R 4.3.0)
glmnet	4.1-8	Attached	2023-08-22	CRAN (R 4.3.0)
HIMA	2.2.1	Attached	2023-09-10	CRAN (R 4.3.0)
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)
lavaan	0.6-16	Attached	2023-07-19	CRAN (R 4.3.0)
lubridate	1.9.3	Attached	2023-09-27	CRAN (R 4.3.1)
LUCIDus	2.2.1	Attached	2022-11-08	CRAN (R 4.3.0)
MASS	7.3-60	Attached	2023-05-04	CRAN (R 4.3.1)
Matrix	1.6-1.1	Attached	2023-09-18	CRAN (R 4.3.1)
mclust	6.0.0	Attached	2022-10-31	CRAN (R 4.3.0)
ncvreg	3.14.1	Attached	2023-04-25	CRAN (R 4.3.0)
networkD3	0.4	Attached	2017-03-18	CRAN (R 4.3.0)
nnet	7.3-19	Attached	2023-05-03	CRAN (R 4.3.1)
OpenMx	2.21.8	Attached	2023-04-05	CRAN (R 4.3.0)
plotly	4.10.2	Attached	2023-06-03	CRAN (R 4.3.0)
progress	1.2.2	Attached	2019-05-16	CRAN (R 4.3.0)
purrr	1.0.2	Attached	2023-08-10	CRAN (R 4.3.0)
r.jive	2.4	Attached	2020-11-17	CRAN (R 4.3.0)
readr	2.1.4	Attached	2023-02-10	CRAN (R 4.3.0)
RMediation	1.2.2	Attached	2023-05-12	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)

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

Package	Version	Attached or Loaded	Date/Publication	Source
usethis	2.2.2	Attached	2023-07-06	CRAN (R 4.3.0)
xtune	2.0.0	Attached	2023-06-18	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)
backports	1.4.1	Loaded	2021-12-13	CRAN (R 4.3.0)
BiocGenerics	0.46.0	Loaded	2023-06-04	Bioconductor
bitops	1.0-7	Loaded	2021-04-24	CRAN (R 4.3.0)
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)
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)
ellipsis	0.3.2	Loaded	2021-04-29	CRAN (R 4.3.0)
evaluate	0.22	Loaded	2023-09-29	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)
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)
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)
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)

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

Package	Version	Attached or Loaded	Date/Publication	Source
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)
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)
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)
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)
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)
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)
mix	1.0-11	Loaded	2022-05-31	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)
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)
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)
ps	1.7.5	Loaded	2023-04-18	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	2.32.0	Loaded	2023-05-08	Bioconductor
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)

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

Package	Version	Attached or Loaded	Date/Publication	Source
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)
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	1.2.3	Loaded	2023-01-31	CRAN (R 4.3.0)
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	2.5.1	Loaded	2023-09-26	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)

5.2 *Software Environment*

This book was developed on the following platform.

Setting	Value
version	R version 4.3.1 (2023-06-16)
os	macOS Sonoma 14.0
system	aarch64, darwin20
ui	X11
language	(EN)
collate	en_US.UTF-8
ctype	en_US.UTF-8
tz	America/Los_Angeles
date	2023-10-16
pandoc	3.1.1 @ /Applications/RStudio.app/Contents/Resources/app/quarto/bin/tools/ (via rmarkdown)

6 Supplemental Code

6.1 High Dimensional Mediation Analysis Code

6.1.1 Early Integration of omics datasets

```
#' Conducts High Dimensional Mediation Analysis with Early Integration
#'
#' Given exposure, outcome and multiple omics data,
#' function runs HIMA mediation analysis with early integration
#' and returns a tidy dataframe for the results
#'
#' @param exposure A numeric vector for the exposure variable
#' @param outcome A numeric vector for the outcome variable
#' @param omics A list of numeric matrices representing omics data
#' @param covs A numeric matrix representing the covariates
#'
#' @return A tidy dataframe summarizing the results of HIMA analysis
#'
#' @import dplyr
#' @importFrom tidyr as_tibble
#' @importFrom dplyr left_join
#' @importFrom janitor remove_empty
#' @importFrom purrr map_lgl
#' @importFrom stringr str_detect
#' @importFrom stats gaussian
#' @importFrom HIMA hima
#' @importFrom base as.matrix
hima_early_integration <- function(exposure,
                                   outcome,
                                   omics_lst,
                                   covs,
                                   Y.family = "binomial",
                                   M.family = "gaussian") {
  # Give error if covs is NULL
  if (is.null(covs)) {
    stop("Currently, hima does not support analysis without covariates.
         Please provide covariates.")
  }
```

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

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))

# Create data frame of omics data
omics_df <- omics_lst_df  %>%
  purrr::reduce(left_join, by = "name") %>%
  column_to_rownames("name")

# Run hima
result_hima_early <- hima(X = exposure,
                          Y = outcome,
                          M = omics_df,
                          COV.XM = covs,
                          COV.MY = covs,
                          Y.family = Y.family,
                          M.family = M.family,
                          verbose = FALSE,
                          max.iter = 100000,
                          scale = FALSE) %>%
  as_tibble(rownames = "ftr_name")

# Reorders the columns and adds the omics layer information
result_hima_early <- result_hima_early %>%
  dplyr::mutate(
    multiomic_mthd = "Early Integration",
    mediation_mthd = "HIMA") %>%
  dplyr::select(multiomic_mthd, mediation_mthd,
                ftr_name,
                everything())
# Filter to significant features only and scale % total effect to 100
result_hima_early <- result_hima_early %>%
  filter(BH.FDR < 0.05) %>%
  mutate(pte = 100*`% total effect`/sum(`% total effect`),
         sig = if_else(BH.FDR < 0.05, 1, 0)) %>%
  rename(ie = 'alpha*beta',
         `TE (%)` = pte)
```

```
  # Merge results with feature metadata
  result_hima_early <- result_hima_early %>%
    left_join(meta_df, by = "ftr_name")

  # Return result
  return(result_hima_early)
}
```

6.1.2 Intermediate Integration of omics datasets

```
#' Conducts High Dimensional Mediation Analysis with Intermediate Integration
#' Combines -omic data with covariates to calculate the indirect effect
#' of each individial, possible mediating feature on the relationship
#' between exposure and outcome using cooperative group lasso
#'
#' @param omics list of dataframes with -omic data
#' @param covs dataframe with covariate data
#' @param outcome vector with outcome variable data
#' @param exposure vector with exposure variable data
#' @param n_boot number indicating number of bootstrap estimates to perform for se
#'
#' @return dataframe with the following variables: ftr_name, omic_layer,
#' alpha, alpha_se, s1, beta_bootstrap, beta_se, indirect, ind_effect_se,
#' lcl, ucl, gamma, pte_intermediate, sig_intermediate
#'
#' @examples
#' calculate_mediation(omics = list(omics_1, omics_2),
#'                     covs = covariate_data,
#'                     outcome = outcome_data,
#'                     exposure = exposure_data.
#'                     n_boot = 100)
#'
#' @importFrom epiomics owas
#' @importFrom dplyr bind_cols group_by inner_join mutate
#' @importFrom dplyr select rename filter summarise
#' @importFrom purrr map map2 reduce
#' @importFrom broom tidy
#' @importFrom matrixStats colAnyNA rowAnyNA
#' @importFrom RMediation medci
#' @importFrom boot boot detectCores
#' @importFrom glmnet::groupedlasso grouped.lasso::groupedlasso
#' @importFrom glmnet::cv.groupedlasso cv.groupedlasso::cv.groupedlasso
#' @importFrom glmnet::cvglmnet cvglmnet::cvglmnet
#' @importFrom glmnet::predict.cv.glmnet predict.cv.glmnet
```

```r
#' @importFrom matrixStats colAlls rowSds rowMeans
#' @importFrom stringr str_detect
hima_intermediate_integration <- function(exposure,
                                          outcome,
                                          omics_lst,
                                          covs = NULL,
                                          n_boot,
                                          Y.family = "gaussian") {
  ## Change omics elements to dataframes
  omics_lst_df <- purrr::map(omics_lst, ~as_tibble(.x, rownames = "name"))

  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))

  ## Create data frame of omics data
  omics_df <- omics_lst_df  %>%
    purrr::reduce(left_join, by = "name") %>%
    column_to_rownames("name")

  # Rename family for xtune function
  if(Y.family != "gaussian") {stop("Only continuous outcomes currently supported")}

  # Get dataframe of all data
  full_data <- tibble(outcome = outcome,
                      exposure = exposure) %>%
    bind_cols(omics_df)

  # Add covs if not null
  if(!is.null(covs)) {full_data <- full_data %>% bind_cols(covs)}

  # Get external information matrix
  # Convert each data frame to a long format and extract the unique column names
  external_info <- purrr::map(omics_lst,
                              ~data.frame(column = colnames(.x), val = 1)) %>%
    map2(names(omics_lst), ~dplyr::rename(.x, !!.y := val)) %>%
    purrr::reduce(., full_join, by = "column") %>%
    replace(is.na(.), 0) %>%
    column_to_rownames("column")

  # 0) calculate gamma (x --> y) ----
```

```r
if(Y.family == "binary"){
  gamma_est <- coef(glm(outcome ~ exposure + as.matrix(covs),
                        family = binomial))[["exposure"]]
} else if(Y.family == "gaussian"){
  gamma_est <- coef(lm(outcome ~ exposure ))[["exposure"]]
}

# 1) X --> M ----------
# Model 1: x --> m
x_m_reg <- epiomics::owas(df = full_data,
                          omics = rownames(external_info),
                          covars = colnames(covs),
                          var = "exposure",
                          var_exposure_or_outcome = "exposure") %>%
  dplyr::select(feature_name, estimate, se) %>%
  dplyr::rename(alpha = estimate,
                alpha_se = se)

# 2) M--> Y: select features associated with the outcome using group lasso ----
# x+M-->Y Glasso
X = as.matrix(full_data[, colnames(omics_df)])
Y = full_data$outcome
Z = as.matrix(external_info)
U = as.matrix(full_data["exposure"])
if(is.null(covs)){
  U = as.matrix(full_data["exposure"])
} else {
  U = as.matrix(full_data[,c(colnames(covs), "exposure")])
}

# Run xtune
invisible(
  capture.output(
    xtune.fit_all_data <- xtune(X = X, Y = Y, Z = Z, U = U,
                                c = 1,
                                family = "linear")
  )
)

# Extract estimates
xtune_betas_all_data <- as_tibble(as.matrix(xtune.fit_all_data$beta.est),
                                  rownames = "feature_name") %>%
  left_join(meta_df, by = c("feature_name" = "ftr_name")) %>%
  dplyr::filter(feature_name %in% colnames(omics_df))
```

```r
# 3) Calculate SE for Model 3: x+m to y reg -----
# 3.1) boot function ------------------------------------------------------
group_lasso_boot <- function(data, indices, external_info, covs = NULL) {
  X = as.matrix(data)[indices, rownames(external_info)]
  Y = data$outcome[indices]
  if(is.null(covs)){
    U = as.matrix(data[indices,"exposure"])
  } else {
    U = as.matrix(data[indices,c(colnames(covs), "exposure")])
  }
  # Run xtune with error catching function, sometimes xtune needs to run again
  success <- FALSE
  attempts <- 0
  while(!success & attempts < 10) {
    tryCatch({
      # Run xtune
      xtune.fit <- xtune(X = X, Y = Y, Z = as.matrix(external_info), U = U,
                         sigma.square = estimateVariance(X,Y),
                         c = 0,
                         family = "linear", message = FALSE)


      # If the xtune call is successful, proceed with the rest of the code
      # Select betas, drop intercept
      xtune_betas <- as_tibble(as.matrix(xtune.fit$beta.est),
                               rownames = "feature_name") %>%
        dplyr::filter(feature_name %in% colnames(omics_df)) %>%
        dplyr::select(s1) %>%
        as.matrix()
      # Fix issue where sometimes lasso returns a null matrix
      if(sum(dim(xtune_betas) == c(nrow(external_info), 1))==2){
        return(xtune_betas)
      } else {
        return(as.matrix(rep(0, nrow(external_info))))
      }

      success <- TRUE
    }, error = function(e) {
      attempts <- attempts + 1
      print(paste0("Encountered error: ", e$message))
      if (attempts < 10) {
        print("Retrying...")
      } else {
        print("Maximum number of attempts reached. Exiting...")
        return(NULL)
```

```
      }
    })
  }
}


# 3.2) Run Bootstrap analysis ----------------------
# Run Bootstrap
if(is.null(covs)){
  boot_out <- boot(data = full_data,
                   statistic = group_lasso_boot,
                   R = n_boot,
                   # strata = as.numeric(full_data$h_cohort),
                   ncpus = detectCores(),
                   parallel = "multicore",
                   external_info = external_info)
} else {
  boot_out <- boot(data = full_data,
                   statistic = group_lasso_boot,
                   R = n_boot,
                   # strata = as.numeric(full_data$h_cohort),
                   ncpus = detectCores(),
                   parallel = "multicore",
                   external_info = external_info,
                   covs = covs)
}


# Calculate percent of times feature was selected
glasso_boot_results <- tibble(
  feature_name = rownames(external_info),
  beta_bootstrap = colMeans(replace_na(boot_out$t, 0)),
  beta_se = apply(replace_na(boot_out$t, 0), 2, sd)) %>%
  left_join(meta_df, by = c("feature_name" = "ftr_name"))

# 3.4) Join unpenalized results with glasso results ----
int_med_coefs <- dplyr::inner_join(xtune_betas_all_data,
                                   glasso_boot_results,
                                   by = c("feature_name",
                                          "omic_layer", "omic_num")) %>%
  dplyr::inner_join(x_m_reg, by = "feature_name")

# Calculate confidence intervals -----
# mu.x: a1 from reg m = a0 + a1*X
# mu.y: b2 from reg y = b0 + b1*X + b2*M
```

```
  int_med_res <- int_med_coefs %>%
    group_by(feature_name) %>%
    nest() %>%
    mutate(res = purrr::map(data,
                            ~RMediation::medci(mu.x = .x$alpha,
                                               se.x = .x$alpha_se,
                                               mu.y = .x$beta_bootstrap,
                                               se.y = .x$beta_se,
                                               type = "dop") %>%
                              unlist() %>% t() %>% as_tibble())) %>%
    unnest(c(res, data)) %>%
    ungroup() %>%
    janitor::clean_names()

  # Modify results
  intermediate_int_res <- int_med_res %>%
    janitor::clean_names() %>%
    rename(indirect = "estimate",
           ind_effect_se = "se",
           lcl = x95_percent_ci1,
           ucl = x95_percent_ci2) %>%
    mutate(gamma = gamma_est,
           pte = (indirect)/gamma,
           sig = if_else(lcl>0|ucl<0, 1, 0))

    # Filter to significant features only and scale % total effect to 100
  intermediate_int_res <- intermediate_int_res %>%
    filter(sig == 1) %>%
    mutate(pte = 100*pte/sum(pte))


  # Rename feature name
  intermediate_int_res <- intermediate_int_res %>%
    dplyr::rename(ftr_name = feature_name,
                  ie = indirect,
                  beta = beta_bootstrap,
                  `TE (%)` = pte)

  return(intermediate_int_res)
}
```

6.1.3 Late Integration of omics datasets

```r
#' Conducts High Dimensional Mediation Analysis with Late Integration
#' Performs HIMA mediation analysis on multiple omics layers with late integration
#'
#' This function uses HIMA (High-dimensional Mediation Analysis) to perform
#' mediation analyses on multiple omics layers. For each omics layer, it runs
#' HIMA with exposure, outcome, covariates, and that specific omics layer as
#' input, and then collects the results as a tibble. The results for each type
#' of omics layer are stored in a list and then concatenated to form the final
#' result. The values are scaled to a percentage of total effect and metadata
#' is joined to fill in missing information.
#'
#' @param exposure a numeric vector containing exposure measurements
#' @param outcome a numeric vector containing outcome measurements
#' @param omics a named list containing multiple omics layers
#' @param covs a data frame containing covariate information
#' @param omics_names a data frame with metadata for each omics layer
#' @return a tibble with High-dimensional Mediation Analysis results, including metadata
#' @importFrom dplyr mutate select left_join across
#' @importFrom janitor remove_empty
#' @importFrom HIMA hima
#' @importFrom purrr map
#' @importFrom stats var
#'
#' @export
hima_late_integration <- function(exposure,
                                  outcome,
                                  omics_lst,
                                  covs,
                                  Y.family,
                                  M.family = "gaussian") {

  # Give error if covs is NULL
  if (is.null(covs)) {
    stop("Currently, hima does not support analysis without covariates.
         Please provide covariates.")
  }

  # Get number of omics layers
  n_omics <- length(omics_lst)
  omics_name <- names(omics_lst)

  # Meta data
  omics_lst_df <- purrr::map(omics_lst, ~as_tibble(.x, rownames = "name"))
```

```r
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))
# Start the computation
result_hima_late <- vector(mode = "list", length = n_omics)
for(i in 1:n_omics) {
  # Run HIMA with input data
  result_hima_late[[i]] <- hima(X = exposure,
                                Y = outcome,
                                M = omics_lst[[i]],
                                COV.XM = covs,
                                Y.family = Y.family,
                                M.family = M.family,
                                max.iter = 100000,
                                scale = FALSE) %>%
    as_tibble(rownames = "ftr_name")
}

# Assign omic names
names(result_hima_late) <- names(omics_lst)

# Concatenate the resulting data frames
result_hima_late_df <- bind_rows(result_hima_late, .id = "omic_layer")

# Add key details
result_hima_late_df <- result_hima_late_df %>%
  dplyr::mutate(
    multiomic_mthd = "Late Integration",
    mediation_mthd = "HIMA") %>%
  dplyr::select(multiomic_mthd, mediation_mthd,
                omic_layer, ftr_name,
                everything())

# Filter to significant features only and scale % total effect to 100
result_hima_late_df <- result_hima_late_df %>%
  filter(BH.FDR < 0.05) %>%
  mutate(pte = 100*`% total effect`/sum(`% total effect`),
         sig = if_else(BH.FDR < 0.05, 1, 0)) %>%
  rename(ie = 'alpha*beta',
         `TE (%)` = pte)
```

```
  # Return the final table
  return(result_hima_late_df)
}
```

6.1.4 Plot HIMA

```
#' Plot of High Dimensional Mediation Analysis
#'
#' Given a tidy dataframe summarizing the results of HIMA analysis
#'
#' @param result_hima  a tidy dataframe summarizing the results of HIMA analysis
#'
#' @return a figure of the result of HIMA analysis
#'
#' @import dplyr
#' @importFrom ggplot2 ggplot
#'
plot_hima <- function(result_hima) {
  # Pivot longer for figure
  result_hima_long <- result_hima %>%
    rename(Alpha = alpha,
           Beta = beta) %>%
    pivot_longer(cols = c(Alpha, Beta,`TE (%)`),
                 names_to = "name") %>%
    mutate(name = factor(name, levels = c("Alpha", "Beta", "TE (%)")))

  # Plot features
  p <- ggplot(result_hima_long,
         aes(x = fct_inorder(ftr_name),
             y = value,
             fill = omic_layer)) +
    geom_bar(stat = "identity") +
    facet_grid(name ~ omic_layer,
               scales = "free",
               space = "free_x") +
    scale_fill_brewer(type = "qual", palette = 2) +
    geom_hline(yintercept = 0, linetype = 1, color = "grey50") +
    ylab(NULL) + xlab(NULL) +
    theme(
      strip.background = element_blank(),
      strip.text.x = element_blank(),
      axis.text.x = element_text(angle = 90, hjust = 1, vjust = .5),
      legend.title = element_blank(),
      legend.position = "bottom", # Place the legend at the bottom
```

```
      legend.justification = c(1, 0))

  return(p)
}
```

6.2 Mediation with Latent Factors Analysis Code

6.2.1 Early Integration of omics datasets

```
#' Conduct principal component analysis (PCA) as a dimensional reduction step
#' and selected the top i principal components which explained >80% of the variance.
#' Following the joint dimensional reduction step, conduct mediation analysis.

#' Given exposure, outcome and multiple omics data,
#' function runs HIMA mediation analysis with latent factors in early integration
#' and returns a list including two tidy dataframes for the results
#'
#' @param exposure A numeric vector for the exposure variable
#' @param outcome A numeric vector for the outcome variable
#' @param omics_lst A list of numeric matrices representing omics data
#' @param covs A numeric matrix representing the covariates
#'
#' @return A list including two tidy dataframes summarizing the results of HIMA analysis
#' and one vector indicating integration type
#' one dataframe including the significant result of HIMA analysis with PCs as mediators.
#' another dataframe including the result of feature correlation with significant PCs.
#'
#' @import dplyr
#' @importFrom tidyr as_tibble
#' @importFrom dplyr left_join
#' @importFrom purrr map
#' @importFrom stats prcomp cor
#' @importFrom HIMA hima
#' @importFrom base cumsum min sum scale apply
#' @importFrom stringr str_replace
#'
med_lf_early <- function(exposure,
                         outcome,
                         omics_lst,
                         covs,
                         Y.family = "gaussian",
                         M.family = "gaussian",
                         fdr.level = 0.05) {
```

```r
# Give error if covs is NULL
if (is.null(covs)) {
  stop("Currently, hima does not support analysis without covariates.
       Please provide covariates.")
}

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

# Meta data
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))

# i. Obtain PCs----
omics_df_pca <- prcomp(omics_df, center = TRUE, scale. = TRUE)

# Calculate variance and proportion of variance explained by each PC
vars <- apply(omics_df_pca$x, 2, var)
props <- vars / sum(vars)
cum_props <- cumsum(props)

# Determine the number of PCs needed to explain > 80% of the total variance
n_80_pct <- min((1:length(cum_props))[cum_props > 0.8])

# The first PCs which explain >80% of the variance are used as latent mediators
PCs <- omics_df_pca$x[, 1:n_80_pct] %>% scale()

# ii. Perform HIMA with PCs as mediator----

result_hima_comb_pc <- hima(X  = exposure,
                            Y = outcome,
                            M = PCs,
                            COV.XM = covs,
                            COV.MY = covs,
                            Y.family =  Y.family,
                            M.family = M.family,
                            scale = FALSE)


# Change to tibble and select significant PCs
```

```
  result_hima_pca_early <- as_tibble(result_hima_comb_pc, rownames = "lf_num") %>%
    filter(BH.FDR < fdr.level)

  # Filter Significant PCs, Create scaled %TE variable
  result_hima_pca_early_sig <- result_hima_pca_early %>%
    mutate(
      te_direction = if_else(beta<0, -1*`% total effect`, `% total effect`),
      `% Total Effect scaled` = 100*`% total effect`/sum(`% total effect`) %>%
        round(1),
      multiomic_mthd = "Early") %>%
    mutate(lf_named = str_replace(lf_num, "PC", "Joint Comp. "),
           lf_ordered = forcats::fct_reorder(lf_named, `te_direction`)) %>%
    rename(Alpha = alpha,
           Beta = beta,
           `TE (%)` = `% Total Effect scaled`) %>%
    mutate(omic_num = case_when(str_detect(lf_num, "meth") ~ 1,
                                str_detect(lf_num, "transc") ~ 2,
                                str_detect(lf_num, "miR") ~ 3,
                                str_detect(lf_num, "pro") ~ 4,
                                str_detect(lf_num, "met") ~ 5,
                                TRUE ~ 0))


  # ii correlation of features vs PC's --------------
  # Extract variable correlation with principal components
  var.cor <- cor(omics_df, omics_df_pca$x)

  # Select only significant PCs
  ftr_cor_sig_pcs <- var.cor[,(colnames(var.cor) %in%
                                 result_hima_pca_early_sig$lf_num)]

  ftr_cor_sig_pcs_df <- ftr_cor_sig_pcs %>%
    as_tibble(rownames = "feature") %>%
    left_join(meta_df, by = c("feature" = "ftr_name"))


  res = list(result_hima_pca_early_sig = result_hima_pca_early_sig,
             result_ftr_cor_sig_pcs_early = ftr_cor_sig_pcs_df,
             integration_type = " Early")
  return(res)
}
```

6.2.2 Intermediate Integration of omics datasets

```
#' Perform a joint dimensionality reduction step
#' using Joint and Individual Variance Explained (JIVE).
#' Following the joint dimensionality reduction step,
#' conduct mediation analysis.


#' Given exposure, outcome and multiple omics data,
#' function runs HIMA mediation analysis with latent factors in intermediate integration
#' and returns a list including two tidy dataframes for the results
#'
#' @param exposure A numeric vector for the exposure variable
#' @param outcome A numeric vector for the outcome variable
#' @param omics_lst A list of numeric matrices representing omics data
#' @param jive.rankJ Number of joint factors for JIVE to estimate. If NULL,
#' then jive estimates the optimum number of joint factors.
#' @param jive.rankA Number of individual factors for JIVE to estimate. If NULL,
#' then jive estimates this variable. Should be a numeric vector with the same
#' length as dim(omics_lst)
#' @param covs A numeric matrix representing the covariates
#'
#' @return A list including two tidy dataframes summarizing the results of HIMA analysis
#' and one vector indicating integration type.
#' one dataframe including the significant result of
#' HIMA analysis with latent factors as mediators.
#' another dataframe including the result of feature correlation with significant PCs.
#'
#' @import dplyr
#' @importFrom tidyr as_tibble
#' @importFrom dplyr left_join
#' @importFrom r.jive jive
#' @importFrom stats prcomp cor
#' @importFrom HIMA hima
#' @importFrom base min sum scale apply svd diag
#' @importFrom stringr str_replace
#'
med_lf_intermediate <- function(exposure,
                                outcome,
                                omics_lst,
                                covs,
                                jive.rankJ = NULL,
                                jive.rankA = NULL,
                                Y.family = "gaussian",
                                M.family = "gaussian",
```

```
                              fdr.level = 0.05) {

# Give error if covs is NULL
if (is.null(covs)) {
  stop("Currently, hima does not support analysis without covariates.
       Please provide covariates.")
}

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

# Meta data
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))
# Transpose omics matrices
omics_t <- lapply(omics_lst, t)

# Rename omics datasets
names(omics_t) = names(omics_lst)

# Run JIVE-----
if (is.null(jive.rankJ) | is.null(jive.rankA)) {
  jive.message <- "Step A) Starting JIVE analysis..."
} else {jive.message <- "Step A) Starting JIVE analysis with ranks given..."}
message(paste0(jive.message, "    (",
               Sys.time() %>%
                 format("%H:%M:%S") %>%
                 str_split(":") %>%
                 unlist() %>%
                 .[1:3] %>%
                 paste(collapse = ":"),
               ")\n"))

result_jive2 <- jive(data = omics_t,
                     rankJ = jive.rankJ,
                     rankA = jive.rankA,
                     method = "given",
                     conv = 1e-04,
                     maxiter = 100,
                     showProgress = FALSE)
```

```r
# Get the components from JIVE
# Function to extract joint and individual PCs ----
get_PCs_jive <- function (result){
  # Number of joint factors
  n_joint = result$rankJ
  # Number of individual factors
  n_indiv = result$rankA
  # Get number of data matrices in result
  l <- length(result$data)
  # Calculate total number of PCs to compute
  nPCs = n_joint + sum(n_indiv)
  # Initialize matrix to hold PC scores
  PCs = matrix(nrow = nPCs, ncol = dim(result$data[[1]])[2])
  # Initialize vector to hold PC names
  PC_names = rep("", nPCs)
  # If joint structure is present
  if (n_joint > 0) {
    # Compute SVD on joint structure
    SVD = svd(do.call(rbind, result$joint), nu = n_joint, nv = n_joint)
    # Compute PC scores for joint structure
    PCs[1:n_joint,] = diag(SVD$d)[1:n_joint, 1:n_joint] %*% t(SVD$v[, 1:n_joint])
    # Assign names to joint PCs
    PC_names[1:n_joint] = paste("Joint ", 1:n_joint)
  }
  # Loop over data matrices
  for (i in 1:l) {
    # If individual structure is present for this matrix
    if (n_indiv[i] > 0) {
      # Compute SVD on individual structure
      SVD = svd(result$individual[[i]], nu = n_indiv[i],
                nv = n_indiv[i])
      # Get indices for PCs corresponding to this data matrix
      indices = (n_joint + sum(n_indiv[0:(i - 1)]) + 1):(n_joint +
                                                           sum(n_indiv[0:i]))
      # Compute PC scores for individual structure
      PCs[indices, ] = diag(SVD$d)[1:n_indiv[i], 1:n_indiv[i]] %*%
        t(SVD$v[, 1:n_indiv[i]])
      # Assign names to individual PCs
      PC_names[indices] = paste0(names(result$data)[i],
                                 "_", 1:n_indiv[i])
    }
  }
  # Rename PCs
  rownames(PCs) <- PC_names %>%
    str_replace("  ", "_")
```

```
  # Transpose and change to data.frame
  out <- as.data.frame(t(PCs))
  # Return output
  return(out)
}

factors_jive <- get_PCs_jive(result_jive2) %>%
  dplyr::mutate(across(everything(), ~as.vector(scale(.))))

# run mediation analysis------
message(paste0("Step B) Starting hima on JIVE factors...    (",
               Sys.time() %>%
                 format("%H:%M:%S") %>%
                 str_split(":") %>%
                 unlist() %>%
                 .[1:3] %>%
                 paste(collapse = ":"),
               ")\n"))

# Select only HH:MM:SS from Sys.time()



result_hima_jive <- hima(X = exposure,
                         Y = outcome,
                         M = factors_jive,
                         COV.MY = covs,
                         COV.XM = covs,
                         Y.family = c("gaussian"),
                         M.family = c("gaussian"),
                         verbose = FALSE,
                         scale = FALSE)

# Modify and filter significant
result_hima_jive_1 <- result_hima_jive %>%
  rownames_to_column("lf_num") %>%
  mutate(multiomic_mthd = "Intermediate",
         ind_joint = str_split_fixed(lf_num, fixed("_"), 2)[,1],
         ind_joint_num = case_when(ind_joint == "Joint" ~ 1,
                                   ind_joint == "methylome" ~ 2,
                                   ind_joint == "transcriptome" ~ 3)) %>%
  dplyr::select(multiomic_mthd, everything())

# Filter significant components and create scaled %TE variable
result_hima_jive_sig <-  result_hima_jive_1 %>%
```

```
    filter(BH.FDR<fdr.level) %>%
    mutate(`% Total Effect scaled` =
             round(100*`% total effect`/sum(`% total effect`),1),
           te_direction = if_else(beta < 0,
                                  -1 * `% total effect`,
                                  `% total effect`)) %>%
    mutate(lf_named = str_replace(toTitleCase(lf_num), "_", " Comp. "),
           lf_ordered = forcats::fct_reorder(lf_named, `te_direction`)) %>%
    rename(Alpha = alpha,
           Beta = beta,
           `TE (%)` = `% Total Effect scaled`)%>%
    mutate(omic_num = case_when(str_detect(lf_num, "meth") ~ 1,
                                str_detect(lf_num, "transc") ~ 2,
                                str_detect(lf_num, "miR") ~ 3,
                                str_detect(lf_num, "pro") ~ 4,
                                str_detect(lf_num, "met") ~ 5,
                                TRUE ~ 0))

  # correlation of features vs JIVE factors --------------
  # Extract variable correlation with JIVE Factors
  var.cor <- cor(omics_df, factors_jive)

  # Select only significant PCs
  ftr_cor_sig_pcs_jive <- var.cor[,(colnames(var.cor) %in%
                                      result_hima_jive_sig$lf_num)]

  ftr_cor_sig_pcs_jive_df <- ftr_cor_sig_pcs_jive %>%
    as_tibble(rownames = "feature") %>%
    left_join(meta_df, by = c("feature" = "ftr_name"))

  res = list(result_hima_jive_sig = result_hima_jive_sig,
             result_ftr_cor_sig_pcs_jive = ftr_cor_sig_pcs_jive_df,
             integration = "Intermediate")
  return(res)
}
```

6.2.3 Late Integration of omics datasets

```
#' Conduct 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 joint dimensionality reduction step, conduct mediation analysis.

#' Given exposure, outcome and multiple omics data,
```

```r
#' function runs HIMA mediation analysis with latent factors in late integration
#' and returns a list including two tidy dataframes for the results
#'
#' @param exposure A numeric vector for the exposure variable
#' @param outcome A numeric vector for the outcome variable
#' @param omics_lst A list of numeric matrices representing omics data
#' @param covs A numeric matrix representing the covariates
#'
#' @return A list including two tidy dataframes and one vector.
#' Two tidy dataframes summarized the results of HIMA analysis
#' one dataframe including the significant result of
#' HIMA analysis with PCs as mediators for each omics layer.
#' another dataframe including the result of feature correlation with significant PCs.
#' One vector includes the integration type.
#'
#' @import dplyr
#' @importFrom tidyr as_tibble
#' @importFrom dplyr left_join
#' @importFrom purrr map2 mapreduce
#' @importFrom stats prcomp cor
#' @importFrom HIMA hima
#' @importFrom base cumsum min sum scale apply
#' @importFrom stringr str_replace
#'
med_lf_late <- function(exposure,
                        outcome,
                        omics_lst,
                        covs,
                        Y.family = "gaussian",
                        M.family = "gaussian",
                        fdr.level = 0.05) {
  # Give error if covs is NULL
  if (is.null(covs)) {
    stop("Currently, hima does not support analysis without covariates.
         Please provide covariates.")
  }

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

  # Meta data
  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))

# Define function to run PCA on a matrix and return loadings and scores
run_pca <- function(mat, omic_name) {
  # perform PCA on input matrix with scaling
  pca <- prcomp(mat, scale. = TRUE)
  # calculate variance explained by each principal component
  vars <- apply(pca$x, 2, var)
  props <- vars / sum(vars)
  # calculate cumulative proportion of variance explained
  cum_props <- cumsum(props)

  # determine the number of principal components needed to explain 80% of the variance
  n_80_pct <- min((1:length(cum_props))[cum_props > 0.8])
  # create a dataframe of scores for the principal components and scale them
  PCs <- pca$x[, 1:n_80_pct] %>% scale() %>% as.data.frame()
  colnames(PCs) <- paste0(omic_name, "_", colnames(PCs))
  # create a dataframe of loadings for the principal components and scale them
  loadings_df <- pca$rotation[, 1:n_80_pct] %>% scale() %>% as.data.frame()
  colnames(loadings_df) <- paste0(omic_name, "_", colnames(loadings_df))
  loadings_df <- rownames_to_column(loadings_df, "feature")

  # Including all PCs
  # create a dataframe of scores for the principal components and scale them
  PCs_full <- pca$x %>% scale() %>% as.data.frame()
  colnames(PCs_full) <- paste0(omic_name, "_", colnames(PCs_full))

  # create a dataframe of proportion of variance explained by each principal component
  props_df <- data.frame(pc_num = paste0(omic_name, "_",
                                         names(props)),
                         pc_var_explained = props)
  rownames(props_df) <- NULL
  # return a list of results
  return(list(loadings = loadings_df, scores = PCs, scores_full = PCs_full,
              n_pcs_80_pct = n_80_pct, pc_var_explained = props_df))
}
# i. get PCs and HIMA -----
# PCA scores: Apply function to each matrix in the list and collect
scores_list_late_int <-  map2(omics_lst,
                              names(omics_lst),
                              ~run_pca(.x, .y)$scores)
scores_df <- purrr::reduce(scores_list_late_int, cbind) %>% as.data.frame()
```

```r
# Loadings: Apply function to each matrix in the list and collect PC
loadings_list_late_int <- map2(omics_lst,
                               names(omics_lst),
                               ~run_pca(.x, .y)$loadings)
loadings_df <- purrr::reduce(loadings_list_late_int, full_join, by = "feature")

# Number of PCs explain >80%: Get number of PCs for each omic
late_int_pcs_80 <- purrr::map(omics_lst, ~run_pca(.x, NULL)$n_pcs_80_pct) %>%
  bind_rows()

# ii. run HIMA with the current dataset and principal components ----
result_hima_late_integration <- hima(X = exposure,
                                     Y = outcome,
                                     M = scores_df,
                                     COV.MY = covs,
                                     COV.XM = covs,
                                     Y.family = c("gaussian"),
                                     M.family = c("gaussian"),
                                     scale = FALSE)

# Filter significant pcs, create scaled %TE variable
result_hima_late_sig <- result_hima_late_integration %>%
  as_tibble(rownames = "lf_num") %>%
  filter(BH.FDR < 0.05) %>%
  mutate(
    te_direction = if_else(beta<0, -1*`% total effect`, `% total effect`),
    `% Total Effect scaled` = 100*`% total effect`/sum(`% total effect`))

result_hima_late_sig <- result_hima_late_sig  %>%
  mutate(lf_numeric = str_split_fixed(lf_num, fixed("_"), 2)[,2],
         omic_layer = str_split_fixed(lf_num, fixed("_"), 2)[,1] %>%
           str_to_sentence(),
         multiomic_mthd = "Late")%>%
  mutate(omic_layer = str_replace(omic_layer, "Mirna", "miRNA" ),
         omic_pc = str_c(omic_layer, " ", lf_numeric) %>%
           str_replace("PC", "Comp. ")) %>%
  mutate(lf_ordered = forcats::fct_reorder(omic_pc, `te_direction`)) %>%
  dplyr::select(multiomic_mthd, omic_pc, omic_layer, lf_num, lf_ordered,
                alpha, beta, `% Total Effect scaled`) %>%
  rename(Alpha = alpha,
         Beta = beta,
         `TE (%)` = `% Total Effect scaled`)%>%
  mutate(omic_num = case_when(str_detect(lf_num, "meth") ~ 1,
                              str_detect(lf_num, "transc") ~ 2,
                              str_detect(lf_num, "miR") ~ 3,
```

```
                                    str_detect(lf_num, "pro") ~ 4,
                                    str_detect(lf_num, "met") ~ 5))


  # Extract variable correlation with principal components
  scores_full_list_late_int <- map2(omics_lst,
                                    names(omics_lst),
                                    ~run_pca(.x, .y)$scores_full)

  scores_df_full <- purrr::reduce(scores_full_list_late_int, cbind) %>% as.data.frame()

  # Get correlation of omics and PCs by omic layer
  var.cor <- map_df(names(omics_lst), function(type) {
    cor(omics_lst[[type]], scores_full_list_late_int[[type]]) %>%
      as.data.frame()
  })

  # Select only significant PCs
  ftr_cor_sig_pcs_late <- var.cor %>%
    dplyr::select(result_hima_late_sig$lf_num) %>%
    as_tibble(rownames = "feature") %>%
    left_join(meta_df, by = c("feature" = "ftr_name"))

  res = list(result_hima_late_sig = result_hima_late_sig,
             result_ftr_cor_sig_pcs_late = ftr_cor_sig_pcs_late,
             intergration_type = "Late")
  return(res)

}
```

6.2.4 Plot Mediation

```
#' Plot result of mediation analysis with latent factors
#'
#' Given a list including two tidy dataframes and one integration type vector
#' summarizing  mediation analysis result
#' function runs plotting and returns a plot
#'
#' @param med_lf_list A list including two tidy dataframes
#' summarizing the results of HIMA analysis
#' and one vector indicating integration type

#'
#' @return a figure of the result of mediation with latent factors in given integration
```

```r
#'
#' @import dplyr
#' @importFrom dplyr left_join
#' @importFrom ggplot2 ggplot
#' @importFrom base apply
#'

plot_med_lf <- function(med_lf_list) {

  # Panel A Bargraph of mediation effects of latent factors --------------
  med_long <- med_lf_list[[1]]%>%
    pivot_longer(cols = c(Alpha, Beta, `TE (%)`))

  # Plot
  panel_a <- ggplot(med_long, aes(x = lf_ordered, y = value)) +
    geom_bar(stat = "identity", fill = "grey50") +
    geom_hline(yintercept = 0) +
    facet_grid(name ~ omic_num, scales = "free", space = "free_x", switch = "y") +
    ggh4x::facetted_pos_scales(
      y = list(name == "Alpha"  ~ scale_y_continuous(
        limits = c(-.45,.45), breaks = c(-0.4, 0, .4)),
        name == "Beta"   ~ scale_y_continuous(
          limits = c(-.45,.45), breaks = c(-0.4, 0, .4)),
        name == "TE (%)" ~ scale_y_continuous(
          limits = c(-1,55), n.breaks = 4))) +
    theme(axis.title = element_blank(),
          strip.placement = "outside",
          strip.text.x = element_blank(),
          strip.text.y = element_text(angle = 0, size = 9),
          axis.text.x = element_blank(),
          strip.background = element_blank(),
          axis.text.y = element_text(size = 8))

  # Panel B: heatmap of correlation of features vs PC's --------------

  # Select features for plots -------
  ## Select rows with values in the top 10 of their respective columns

  top <- apply(med_lf_list[[2]] %>%
                 select(-omic_layer, -omic_num) %>%
                 janitor::remove_empty(which = "rows") %>%
                 column_to_rownames("feature"), 2,
               function(x) x %in% tail(sort(abs(x)), 10))

  ## Filter selected rows
```

```r
ftr_cor_sig_lf_top_ft <- med_lf_list[[2]][which(rowSums(top) > 0), ]

# Pivot longer
ftr_cor_sig_lf_top_ft_l <-
  ftr_cor_sig_lf_top_ft %>%
  pivot_longer(cols = all_of(med_lf_list[[1]]$lf_num),
               names_to = "lf_num",
               values_to = "Correlation")

# Change features which are in wrong JIVE individual component to zero
if(med_lf_list[[3]] == "Intermediate") {
  ftr_cor_sig_lf_top_ft_l <- ftr_cor_sig_lf_top_ft_l %>%
    mutate(
      in_ind_omic = str_sub(lf_num, 1, 5) == str_sub(omic_layer, 1, 5),
      Correlation = ifelse(in_ind_omic | str_detect(lf_num, "Joint"),
                           Correlation, NA))
}

# Join with long format of mediation effects of latent factors
# to get the ordered latent factor
panel_b_dat_top_ft <- left_join(ftr_cor_sig_lf_top_ft_l,
                                med_long %>%
                                  filter(name == "Alpha") %>%
                                  dplyr::select(lf_num, lf_ordered),
                                by = "lf_num") %>%
  mutate(omic_num2 = case_when(str_detect(lf_num, "meth") ~ 1,
                               str_detect(lf_num, "transc") ~ 2,
                               str_detect(lf_num, "miR") ~ 3,
                               str_detect(lf_num, "pro") ~ 4,
                               str_detect(lf_num, "met") ~ 5,
                               TRUE ~ 0))


# Plot
panel_b <- ggplot(data = panel_b_dat_top_ft,
                  aes(y = feature,
                      x = lf_ordered,
                      fill = Correlation)) +
  geom_tile(color = "white") +
  facet_grid(omic_layer ~ omic_num2, scales = "free", space = "free") +
  scale_fill_gradient2(low  = "blue",
                       mid  = "white",
                       high = "red",
                       midpoint = 0,
                       limits = c(-1, 1),
```

```
                          breaks = c(-1, 0, 1),
                          na.value = "grey50") +
    theme(
      axis.text.x = element_text(size = 8,angle = 90, hjust = 1, vjust = .5),
      axis.text.y = element_text(size = 8),
      strip.text = element_blank(),
      axis.title = element_blank(),
      axis.ticks.x = element_blank(),
      legend.position = "none",
      text = element_text(size = 8))

  # Combine Figures
  p <- cowplot::plot_grid(
    NULL, panel_a,  NULL, panel_b,
    ncol = 1, align = "v", axis = "lr",
    rel_heights  = c(.05, .6, .1, 1.75),
    labels = c("a)","", "b) "))

  return(p)
}
```

6.3 Integrated/Quasi Mediation Analysis Code

6.3.1 Early Integration of omics datasets

```
#' 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
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]

# 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"]]
```

```r
# 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)
}
```

6.3.2 Intermediate Integration of omics datasets

6.3.3 Late Integration of omics datasets

```
# Plot Lucid In Serial Function ------
source(fs::path(dir_proj, "functions", "lucid_reorder_plot_without_y.R"))

# 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$Znames)
  valueGtoX <- as.vector(t(x$pars$beta[, -1]))
  valueXtoZ <- as.vector(t(x$pars$mu))
  valueXtoY <- as.vector(x$pars$gamma$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 #HW added
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)
}


# lucid_fit1 <- fit1
# lucid_fit2 <- fit2
# lucid_fit3 <- fit3

# 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) ----
  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
```

```r
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>"))


## 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)


## 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 ----

# 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$value+.00000000000000000000001,
  color = links_new$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)
}
```

6.3.4 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 integreation,, "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) {
  if(integration_type == "Early"){
    M_mean = as.data.frame(fit$pars$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$res_Mu_Sigma$Mu[[1]], rownames = "variable") %>%
    bind_rows(as_tibble(fit$res_Mu_Sigma$Mu[[2]], rownames = "variable")) %>%
    bind_rows(as_tibble(fit$res_Mu_Sigma$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)
}
```

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.

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.