ChatGPT

Design Proposal: Inter-Tissue Synchronization Analysis Python Package

Introduction and Objectives

This proposal outlines a modular **Python package** for analyzing inter-tissue gene expression coordination changes across cohorts (e.g., age groups). The package replicates and generalizes the framework from *"A computational framework for detecting inter-tissue gene-expression coordination changes with aging"* nature.com . The goal is to provide a reusable pipeline that can ingest multi-tissue transcriptomic data (such as GTEx), construct multi-tissue co-expression networks, identify gene modules, quantify differential connectivity between cohorts, and apply machine learning to discover biomarkers. The design emphasizes **modularity**, **reproducibility**, and **extensibility** to other datasets, phenotypes, and even other omics or species. All components are accessible via both a command-line interface (CLI) and a Python API, facilitating integration into computational biology workflows.

Key Features of the Framework (from Briller *et al.*, 2025):

- Multi-tissue **Weighted Gene Co-expression Network Analysis (WGCNA)** with refined parameters for inter-tissue connections nature.com .
- **Modular Differential Connectivity (MDC)** analysis to find modules with altered connectivity between groups nature.com nature.com .
- **Machine Learning** classifiers (e.g., Random Forest, XGBoost) trained on gene expression and pathway-derived features to predict cohort (e.g., young vs. old) nature.com .
- **Pathway analysis** including GO/KEGG enrichment and single-sample GSEA to interpret biological functions nature.com .

The following sections describe each module of the proposed package, the responsibilities and key classes/functions, and how they interconnect. A suggested folder structure is provided, alongside notes on extending the package to new data types or phenotypes.

Data Ingestion and Preprocessing

This module handles input data reading, normalization, and preparation for network analysis. It is designed to accommodate **GTEx data** as well as custom datasets. Key responsibilities include:

- **Data Loading**: Functions to load expression matrices for multiple tissues. For GTEx, this might parse their standardized formats or use provided APIs. For custom data, users can supply CSV/TSV or HDF5 files. Example class: `DataLoader` with methods `load_gtex(tissue_list, ...)` and `load_custom(filepaths, ...)` to produce a consistent internal representation (e.g., a dictionary of tissue name -> pandas DataFrame of samples×genes).
- **Quality Control & Filtering**: Removing low-quality samples or genes (e.g., low counts or low variance genes). The user can set thresholds for minimal counts/CPM or variance. For multi-tissue analysis, the module can **select top variable genes per tissue** (for example, select up to 5000 most variable genes in each tissue as in Briller *et al.* nature.com) to reduce noise and computational load.
- **Normalization & Transformation**: Apply log2 transformation (especially if input is count data or TPM) and optional normalization (TPM/RPKM or quantile normalization if needed). Ensures all tissues are on comparable scales.
- **Batch Effect Correction**: Integrate methods like ComBat or linear regression to correct for batch effects or experimental confounders across samples. This is critical for GTEx which has known technical variability. The package will allow batch correction **per tissue or across tissues** depending on study design (with caution to avoid information leakage during correction nature.com).

- **Covariate Regression**: Regress out unwanted covariates (sex, RIN, ischemic time, etc.) while preserving the primary variable of interest (e.g., age). For example, in the aging study non-age covariates (batch, sex, ischemic time) were regressed out to retain age-related variation nature.com . A `Preprocessor` class can implement `regress_covariates(expression, covariate_metadata)` using linear models for each gene. This step will be careful to prevent data leakage (e.g., if doing ML, it can perform regression on training data only, etc., controlled by the workflow logic).
- **Output**: The preprocessing yields cleaned, normalized expression matrices for each tissue, and a combined sample metadata table. These can be saved (in a standardized format like Parquet/CSV) in an **output folder** (e.g., `output/data/`) for reuse. Logging will record all filtering steps and any samples removed.

Key Classes/Functions: `DataLoader` , `Preprocessor` with methods like `load_data()` , `normalize()` , `batch_correct()` , `regress_out()` . This module is easily extended – for a new dataset, one can add a new loader method or subclass `DataLoader` if needed (for instance, a loader for a different consortium's data format). Configuration (see **Configuration** below) will specify paths, tissue names, and preprocessing options.

Network Construction (Multi-Tissue Co-expression)

This module constructs **multi-tissue weighted gene co-expression networks** to capture gene–gene correlation structure both within and between tissues. It generalizes WGCNA to a multi-tissue context:

- **Correlation Calculation**: Compute Pearson correlation for every gene pair, including pairs in the **same tissue** and pairs across **different tissues**. If samples are paired by donor across tissues (as in GTEx), the correlation for an inter-tissue gene pair is computed across the donors that have both tissues' expression available. The module will handle subsetting to matched samples for each tissue pair (e.g., using donor IDs from metadata). We provide a function `compute_correlation(data_dict)` that returns a large correlation matrix for all gene–tissue pairs (size N × N, where N = sum of genes across tissues).

- **Adaptive Adjacency (Soft Thresholding)**: Convert correlation to an adjacency matrix using a soft-threshold power (as in WGCNA). Importantly, this package allows **separate powers for intra- vs inter-tissue correlations**, addressing the typically weaker cross-tissue correlations nature.com . For example, parameter `beta_intra` might default to 6 and `beta_inter` to 3 (as found suitable for scale-free topology in the reference study nature.com nature.com). The adjacency calculation follows:

$$a_{i_m,\, j_k} = \begin{cases} |cor(x_i^m,\ x_j^k)|^{\beta_1}, & \text{if } m = k \text{ (same tissue)}; \\ |cor(x_i^m,\ x_j^k)|^{\beta_2}, & \text{if } m \neq k \text{ (different tissues)}. \end{cases}$$

  This refinement to WGCNA ensures that within-tissue gene-gene links and across-tissue links are both represented on appropriate scales nature.com nature.com . The `NetworkBuilder` (or `CoexpressionNetwork`) class will implement this logic in a method `build_adjacency(cor_matrix, beta_intra, beta_inter)`.

- **Scale-Free Topology Check**: The package can evaluate the scale-free criterion by plotting connectivity distributions. Utility functions (e.g., `estimate_beta_parameters()`) could automate selection of `beta_intra` and `beta_inter` by maximizing scale-free fit (R^2) for each network layer nature.com . By default, it can use pre-determined values or those used in literature nature.com , but advanced users can run an optimization procedure (with plots output to `output/network/scale_free_fit.pdf`).

- **Output/Storage**: The network adjacency matrix (or a sparse representation) is stored, and optionally the raw correlation matrix can be saved. Given potentially large N, the adjacency can be stored as a file (e.g., `.csv` or binary with compression). For further steps, the adjacency is used to derive the Topological Overlap Matrix.

Key Classes/Functions: `NetworkBuilder` with `compute_correlation()` and `build_adjacency()` methods; `MultiTissueNetwork` data structure to hold the resulting network (could include attributes like list of genes, their tissue origin, adjacency matrix, etc.). This module is **extensible** to incorporate other correlation measures (e.g., Spearman or partial correlations) or even **multi-omics networks** (by treating different data types as different "tissue layers"). The two-beta mechanism could be generalized to weighting edges between different data layers differently.

Module Identification (Gene Modules Detection)

Once the co-expression network is built, this module identifies **gene modules** – clusters of genes that are highly co-expressed – using hierarchical clustering on a Topological Overlap Matrix (TOM) and then classifies modules as tissue-specific or cross-tissue:

- **TOM Calculation**: Compute the Topological Overlap Matrix from the adjacency. The TOM gives a network-based similarity measure between gene pairs, considering shared neighbors. We will reuse WGCNA's approach: first calculate TOM similarity for all pairs (this can be done via an R function call through rpy2, or via a Python implementation to be developed). A function `calculate_TOM(adjacency)` returns a matrix (or uses memory-efficient algorithms, since for large networks a full matrix is heavy – possibly focusing on clustering directly without storing full N×N TOM).
- **Hierarchical Clustering**: Perform agglomerative clustering on 1 - TOM (dissimilarity). SciPy or scikit-learn's clustering can be used to generate a dendrogram of genes. We then cut this dendrogram to define modules. We will implement the **dynamic tree cut** algorithm (used by WGCNA) for adaptive branch cutting. This could be done by interfacing with the R `dynamicTreeCut` package via rpy2, or by a custom Python translation. Key parameters (min module size, cut height, deepSplit, etc.) will be exposed via the config.
- **Module Definition**: Each module is a set of genes (across all tissues) that cluster together. We define a `Module` class to hold module attributes: member genes, module eigengene (first principal component of expression), etc. After cutting, each gene is assigned a module label. The module eigengene per sample can be computed for downstream analysis if needed.
- **Tissue Specific (TS) vs Cross-Tissue (CT) Modules**: After modules are defined, classify them based on gene composition. We use the criterion from the reference: if a module contains genes from **multiple tissues with each contributing >5%** of module genes, it is a **cross-tissue (CT)** module; otherwise, it's **tissue-specific (TS)** nature.com . (This effectively used a 95% single-tissue membership cutoff nature.com .) The package will implement this check in a function `classify_module(module)` . The output is a list of modules with a flag indicating CT or TS.

- **Output**: A summary of modules (count, size, tissue composition) is saved (e.g., `output/network/modules.csv`). Optionally, module membership can be saved as a .gmt or .csv (gene-to-module mapping). This helps with later enrichment analysis. Plots such as the **gene dendrogram with module colors** and **TOM heatmaps** for modules will be generated for visualization (see **Visualization** section).

Key Classes/Functions: `ModuleDetector` class with `cluster_genes(adjacency)` and `define_modules(dendrogram)`; `Module` data class (attributes: genes, tissues, label, eigengene, etc.). This module is inherently extensible – adjusting clustering parameters or using alternative community detection algorithms (e.g., Louvain clustering on the network) can be done by modifying this component without affecting others. For other phenotypes or species, the module detection process remains the same, as it's data-agnostic.

Differential Connectivity Analysis (MDC Calculation)

This module quantifies and tests changes in network connectivity of modules between cohorts (e.g., young vs old). It implements the **Modular Differential Connectivity (MDC)** metric and statistical significance testing:

- **Network Inputs**: We assume the dataset is divided into two groups (e.g., old vs young). The co-expression analysis may be repeated for each group separately to get group-specific adjacency matrices, or we use the global network structure with group-specific correlations. In practice, we will compute the adjacency (or correlation) for each group on the *same set of genes/modules*. (One approach: identify modules on the combined network, then calculate connectivity within those gene sets in each group's data.)
- **MDC Computation**: For each module (set of N genes), compute its **average intra-module connectivity** in group 1 and in group 2. Connectivity here can be defined as the sum of adjacency weights between all gene pairs within the module nature.com . If K_{ij}^{x} is the adjacency between genes i and j in network x, then:

$$MDC(\text{module}, x, y) = \frac{\sum_{i<j \in \text{module}} K_{ij}^{x}}{\sum_{i<j \in \text{module}} K_{ij}^{y}},$$

as described in Briller *et al.* nature.com . By convention, let x be the younger cohort and y the older, so an MDC > 1 means the module's genes are more tightly connected in young (Gain of Connectivity, GOC) and MDC < 1 indicates a Loss of Connectivity (LOC) with age nature.com nature.com . A function `compute_MDC(module, adj_group1, adj_group2)` will perform this calculation.

- **Statistical Testing**: To assess significance, the module implements a permutation test. One method (as per the paper) is to permute gene labels or sample labels to generate a null distribution of MDC for each module nature.com . For example, randomly shuffle the module assignment or recompute connectivity after shuffling expression values between groups. The module will perform, say, 1000 permutations to get an empirical p-value for the observed MDC. P-values are adjusted to control FDR (e.g., Benjamini-Hochberg). Modules with FDR < 0.05 are considered to have significant differential connectivity. These are labeled as significant GOC or LOC modules accordingly.
- **Output**: Results can be summarized as a table `output/differential_connectivity/mdc_results.csv` listing each module, MDC value, p-value, FDR, and classification (GOC/LOC). Additionally, a **MDC plot** can be generated (e.g., scatterplot of MDC for each module, or a bar plot per module). For example, the package can produce a bar chart of MDC values with confidence intervals, highlighting significant ones.

Key Classes/Functions: `DifferentialAnalyzer` with methods `compute_mdc_all(modules, group1_data, group2_data)` and `permutation_test(module)` . By designing this generally, users can plug in **any two cohorts** (e.g., disease vs control, male vs female) to compare network connectivity. If extended to more than two groups, pairwise MDC or an ANOVA-like extension could be implemented in the future. The modular design ensures that replacing the connectivity metric or significance test (for example, using z-statistics from WGCNA's module preservation stats) can be done within this component.

Feature Selection and Machine Learning

This module provides tools to build predictive models (classifiers or regressors) using features derived from the network analysis – either gene expression of selected genes or higher-level features like pathway scores. It also encompasses feature selection techniques to identify the most informative biomarkers:

- **Feature Assembly**: After module analysis, the user can choose feature sets. Common choices:
  - **Key Genes**: e.g., all genes from significant cross-tissue modules (as potential inter-tissue biomarkers). In the aging use-case, genes from GOC modules were of particular interest nature.com . The package can take the union of genes in significant CT modules or allow the user to specify a gene list.
  - **Module Eigengenes**: Each module's eigengene (the first principal component of its genes' expression) can serve as a feature summarizing that module's activity per sample.
  - **Pathway Scores**: (See next section on pathway analysis – scores like ssGSEA per sample can be included as features for classification.)
- **Train/Test Splitting & CV**: Provide functions for splitting data into train/validation/test sets, stratified by the outcome label. The package will emphasize rigorous evaluation using **nested cross-validation**, as done in the reference study (5-fold outer CV with inner CV for hyperparameter tuning) nature.com nature.com . A `ModelTrainer` class can manage this process: e.g., `ModelTrainer(model, X, y).run_nested_cv(k_outer=5, k_inner=3)` to return performance metrics.
- **LASSO Feature Selection**: Implement LASSO (Least Absolute Shrinkage and Selection Operator) for selecting a sparse set of gene features. Using scikit-learn's `LassoCV` or `LogisticRegressionCV` with L1 penalty, the package can identify which genes have non-zero coefficients. This helps reduce dimensionality and highlight candidates. For instance, LASSO can be run on training data to pick top genes that differentiate young vs old nature.com nature.com . A function `select_features_lasso(X, y)` will output the selected feature list and their coefficients. (The theoretical definition of LASSO's optimization is provided in the documentation for completeness nature.com .)
- **Model Training (RF & XGBoost)**: Provide wrappers for training ensemble models:

  - **Random Forest (RF)** classifier (`sklearn.ensemble.RandomForestClassifier`) – useful for handling many features and providing feature importance scores nature.com .
  - **XGBoost** classifier (`xgboost.XGBClassifier`) – boosted trees that often perform well on structured data and also give feature importance nature.com .
  - Both can be run with hyperparameter tuning. The package will integrate either scikit-learn's GridSearchCV or use the libraries' built-in CV for hyperparameter search (as mentioned, a grid search was used in the study nature.com).
- **Evaluation Metrics**: The module will compute standard metrics: ROC curves and AUC, accuracy, precision, recall, F1, etc., averaging across CV folds nature.com . A function `evaluate_model(predictions, truths)` returns these metrics. ROC curve plotting is handled in the visualization module but can be triggered here.
- **Output**: The results from ML experiments are saved in `output/ml/` . This includes the cross-validation scores, the best model parameters, feature importance rankings, and any selected features. For example, it may save a CSV of top features (genes or pathways) with their importance scores (from LASSO coefficients or RF feature importance). Models themselves can be saved (pickle files) if needed for future prediction.

Key Classes/Functions: `ModelTrainer` , `FeatureSelector` (could encapsulate LASSO logic), and utility functions like `nested_cross_validate()` . By abstracting model training, one can easily **extend to new algorithms** (e.g., SVM or neural networks) by adding a new model option. The module is phenotype-agnostic; predicting age group or disease status is just a matter of providing the appropriate labels. For multi-class problems, the same classes can be reused (with cross-entropy loss, one-vs-rest strategy, etc., as needed). The design encourages exploring different feature sets – for example, one could plug in proteomic pathway scores as features without changing the ML pipeline code.

Pathway Analysis and ssGSEA Projection

This module focuses on **biological interpretation** of the modules and features, by leveraging known gene sets and pathways:

- **Pathway Enrichment (Module Enrichment)**: For a given set of genes (e.g. genes in a module, or a list of top genes), perform over-representation analysis for GO terms, KEGG pathways, Reactome pathways, etc. The package will interface with R's `clusterProfiler` for this purpose nature.com , as it provides convenient functions for GO/KEGG enrichment. For example, `run_enrichment(gene_list, background_genes)` can call an R function that returns significantly enriched GO terms for those genes. The results (enriched pathways with p-values, FDR, gene overlaps) are then parsed back into Python (e.g., as pandas DataFrame). This helps characterize each module (e.g., noting if a cross-tissue module is enriched for "immune response" or "metabolic process"). These results would be saved to `output/enrichment/module_<X>_enrichment.csv` and summarized in reports.
- **Single-Sample GSEA (ssGSEA)**: Implement the single-sample GSEA to compute a **pathway activity score for each sample** nature.com . The package can use a Python implementation (such as the `gseapy` library's ssGSEA function) or call R's GSVA/ssGSEA. Input required is a collection of gene sets (e.g., all Reactome pathways or custom signatures) and the gene expression matrix for each tissue or combined. The function `calculate_ssgsea(expression, gene_sets)` returns a matrix of scores (samples × pathways). These scores are **normalized** per the standard (division by range of values) nature.com . If multiple tissues per individual are present, scores could be computed per tissue or potentially for a combined profile – our design will initially compute scores per tissue dataset.
- **Pathway Score Comparison**: With ssGSEA scores, we can compare pathway activation between groups. For example, test if "cholesterol metabolism" pathways have significantly different scores in young vs old. A function `compare_pathway_scores(scores, labels)` can do t-tests or rank-sum tests for each pathway, with BH-FDR correction nature.com to identify significantly altered pathways (FDR < 0.05). These results reveal systemic differences and align with findings like the **lipid metabolism and immune pathways involvement in aging** nature.com .

- **Pathway-Based Features**: The module also supports using pathway scores as features for ML. For instance, one can train an RF model on the ssGSEA score matrix to predict age. This provides a *lower-dimensional, biologically informed feature space* (as noted in the framework nature.com). The integration with the ML module allows either replacing gene expression features with pathway scores or combining them. The configuration might allow a switch: e.g., `features: pathways` vs `features: genes` to easily toggle this.
- **Output**: Key outputs include pathway enrichment tables and plots (bar graphs of top enriched pathways per module, etc.), and the per-sample pathway score matrix (saved as `output/pathways/ssgsea_scores.tsv`). Significant pathway differences between cohorts can be saved in a summary (like `significant_pathways.csv`). If pathway-based classification is performed, its results are reported similarly to gene-based classification.

Key Classes/Functions: `PathwayAnalyzer` with methods `enrich_modules(modules)` and `compute_ssgsea(expression, gene_sets)` . Internally uses either Python libraries or R integration for enrichment and ssGSEA. Extending this module might involve adding support for other gene set analyses (e.g., **GSVA**, **PLAGE** methods) or incorporating **metabolomic pathways**, etc. Cross-species extension might include using ortholog mapping to apply pathways from one species to another. The modular design lets users plug in their own gene sets (via configuration, e.g., supplying a GMT file of pathways).

Visualization Module

Visualization functions are provided to facilitate interpretation at each step. This module will generate publication-quality plots and interactive visualizations where appropriate:

- **Network and Modules**:
  - *TOM Heatmap*: Plot the topological overlap matrix for each module or a selection of modules. Typically, WGCNA visualizes a TOM heatmap for all genes sorted by module, but for large networks we may do this per module. The heatmap shows the strength of connection between genes, highlighting dense clusters. A function `plot_tom_heatmap(module)` will create such a heatmap (e.g., using matplotlib or seaborn) with genes ordered by the module's dendrogram order.

  - *Module Dendrogram*: Visualize the gene clustering dendrogram with branch cuts. We will output a dendrogram with modules colored. This helps to see the module structure and the cut height. Implemented via scipy's dendrogram function; provided by `plot_dendrogram(genes, linkage_matrix, module_labels)`.
  - *Connectivity Bar Plots*: For each module, visualize connectivity differences. For example, a bar chart could show the **average connectivity** of that module's genes in each group (young vs old) side by side, or a scatter of each gene's connectivity. This corresponds to illustrating MDC results – modules with significant differences can be highlighted. Function `plot_module_connectivity(module, conn_group1, conn_group2)` to produce such a plot.
- **Machine Learning**:
  - *ROC Curves*: Plot ROC curves for the classifiers (average over CV folds or for each fold). Possibly overlay multiple models (LASSO vs RF vs XGB) to compare performance. Provided by `plot_roc_curve(fpr, tpr)` or higher-level `plot_cv_roc(cv_results)` that uses stored CV results.
  - *Feature Importance Plot*: Barplot of top features (genes or pathways) ranked by importance (coefficient magnitude or Gini importance). `plot_feature_importance(feature_importances)` will label the features (gene names or pathway names). This highlights biomarkers like the brain-derived genes identified with high importance in the aging study nature.com nature.com .
  - *Confusion Matrix or Scores*: A heatmap for confusion matrix or a table of precision/recall can be generated for the final model performance.
- **Pathway Visualization**:
  - *Pathway Score Boxplots*: For significant pathways, draw boxplots (or violin plots) of ssGSEA scores in each cohort. This shows, for example, how **lipid metabolism scores differ between young and old** clearly nature.com . `plot_pathway_scores(pathway_name, scores, labels)` will produce this.
  - *Enrichment Bar Chart*: For module enrichment results, a bar chart of top GO terms or pathways (with –log10 p-values) can be created (`plot_enrichment_terms(enrichment_results)`), to quickly view which functions are over-represented.

- **General**: The visualization functions will allow saving to disk (e.g., PNG/PDF files in `output/plots/` with clear names). If used in a Jupyter notebook or via API, they can also return matplotlib Figure objects for interactive display. The design ensures that plots have clear labels, legends, and captions.

Key Classes/Functions: Likely a functional approach is sufficient here (e.g., a `plots.py` module with functions as described). If complexity grows, a `Visualizer` class might manage multi-plot layouts or styling. The module can be extended to include interactive network visualization (e.g., using Plotly or Cytoscape for modules) in the future. By keeping visualization code separate, the analytical core stays uncluttered, and alternative plotting libraries can be integrated with minimal changes.

R Integration (WGCNA & Enrichment)

Some steps of the pipeline rely on algorithms with robust implementations in R. This module provides an interface to those, ensuring the Python package can leverage existing methods until fully equivalent Python implementations are available:

- **WGCNA via rpy2**: Constructing co-expression networks and detecting modules can be done through R's **WGCNA** package nature.com for reliability. The `r_wgcna.py` module will use **rpy2** to call functions like `pickSoftThreshold`, `blockwiseModules` or `adjacency` and `TOMsimilarity`. For example, a function `run_wgcna(expression_data, beta_intra, beta_inter)` might actually call a custom R script that merges multi-tissue data and runs an adapted WGCNA procedure (since WGCNA by default doesn't handle two beta values, we might implement the adjacency calculation in Python and then feed the adjacency into R for TOM and dynamicTreeCut). We will use R mainly for the **dynamic tree cut** and module detection, unless we implement that in Python. This gives us consistency with the original method and faster development.

- **clusterProfiler**: For GO/KEGG enrichment, the package will call R's `clusterProfiler` nature.com because it provides up-to-date gene set libraries and convenient functions for enrichment tests. Via rpy2, we can call `enrichGO`, `enrichKEGG`, etc., and retrieve the results as DataFrames. The integration module will contain the R templates and conversion logic (e.g., ensuring gene IDs match the expected format for GO/KEGG, installing any needed Bioconductor packages on first use).
- **GSVA / ssGSEA**: If needed, the R **GSVA** package (which includes ssGSEA) can be used to calculate pathway scores. Alternatively, the Python `gseapy` can be used – the integration will choose whichever is more straightforward. Ensuring identical results to known implementations is important for validation, so using R's method might be preferred initially.
- **Environment Management**: We will include checks to ensure the R packages are installed. Perhaps a script to install required R packages or documentation to guide the user. The integration functions will be robust to missing R dependencies (throwing a clear error or installing them if possible).
- **Optional Use**: While R integration is powerful, the package is structured such that one could bypass it (for example, by using only Python implementations if they prefer). The design isolates R-dependent code, so in the future these can be swapped out for pure Python functions without affecting the rest of the system.

Key Classes/Functions: Functions like `run_dynamic_tree_cut(dissimilarity_matrix)` or `r_enrich_go(gene_list)` hidden behind Python-friendly APIs (`ModuleDetector` might call these). The **integration module** abstracts away rpy2 details, returning plain Python data structures. This modular approach also means that if a user wants to incorporate another R routine (say, an edge case analysis), they can follow the integration templates here.

Configuration and Reproducibility

To ensure the pipeline is **configurable and reproducible**, the package supports centralized configuration and logging:

- **Configuration Files**: Users can provide a YAML or JSON config file specifying all parameters and file paths. For example, `config.yaml` might contain sections for `data` (list of tissues, file paths, covariates to regress, etc.), `network` (beta_intra, beta_inter, min_module_size, etc.), `ml` (algorithm choices, hyperparameter ranges for grid search), and `pathways` (e.g. gene sets to use, such as "Reactome, GO" or custom GMT file path). A configuration parser (`config.parse_config(file)`) will load these and apply defaults for anything not specified. This makes analyses **reproducible** and easily shareable – the entire pipeline can be rerun on a new dataset by modifying the config rather than code.
- **Default Settings**: The package will include default configurations tuned to replicate the results of the reference study (for aging and GTEx). For instance, default `beta_intra=6, beta_inter=3` nature.com , `min_module_size=30` , `lasso_cv_folds=5` , etc., and references to default gene sets (e.g., use GO/KEGG from clusterProfiler). These defaults are documented, and the config allows overriding them.
- **Logging**: A `logger` (built on Python's `logging` library) records each step of the pipeline with timestamps and parameter values. Logs include summaries of data preprocessing (e.g., "Filtered 200 low-expression genes in Adipose tissue"), network stats ("Chosen beta_intra=6 yields scale-free R^2=0.90"), module counts, and ML performance metrics per fold. The log can be written to a file `output/run.log` for inspection. This ensures transparency in what was done and facilitates debugging.
- **Output Organization**: The package enforces a structured output directory (which can be set in config). For example:

text Copy

```
output/
├── data/                   # preprocessed data, covariate info
├── network/                # network adjacency and module info
├── differential_connectivity/  # MDC results
├── enrichment/             # pathway enrichment results
├── pathways/               # ssGSEA scores or pathway analyses
├── ml/                     # machine learning models, feature lis
└── plots/                  # all figures generated
```

Each subfolder contains well-named files for easy reference. By standardizing this, users know where to find outputs and can write additional scripts to consume these results.

- **Versioning and Reproducibility**: The package will include version tags, and the log will record the package version and possibly git commit if installed from source. Random seeds for any stochastic process (CV splitting, model training randomness, permutations in MDC) can be fixed via config to allow exact reproduction of results. If using multi-threading or parallel processing (for speed in permutation tests or CV), the package will do so deterministically when possible or document the non-deterministic aspects.

Key Classes/Functions: `ConfigManager` for parsing and validating config options, `LoggerSetup` to initialize logging. The emphasis on configuration means the pipeline can be run end-to-end by pointing the CLI to a config file, aiding reproducibility. For extension, new parameters can be added to the config schema as new features are implemented (e.g., adding an option for a new classifier or a new data type).

Command-Line Interface (CLI) and API Access

The package is designed to be used both programmatically (importing its modules in Python) and as a standalone tool via command line:

- **CLI Usage**: A command-line entry point (e.g., `intertissue-sync` or `itsync` as a console script) will be provided. Using a single config file, a user can execute the full pipeline. Example command:

  bash Copy

  ```bash
  itsync run --config config.yaml
  ```

  This would trigger an orchestrated run: loading data, building networks, identifying modules, computing MDC, and running ML and pathway analysis as configured. The CLI could also have subcommands for specific tasks (for instance, `itsync preprocess ...` or `itsync train-model ...`) if the user wants to run steps independently.

- **Interactive API**: All functionality is accessible via Python API for use in notebooks or scripts. For example:

python Copy

```python
from intertissue_sync import DataLoader, NetworkBuilder, ModuleDet
# Load and preprocess data
data = DataLoader(config).load_data()
preprocessed = Preprocessor(config).process(data)
# Build network and detect modules
net = NetworkBuilder(config).build_network(preprocessed)
modules = ModuleDetector(config).find_modules(net)
# Differential connectivity
mdc_results = DifferentialAnalyzer(config).compare_groups(modules,
# Feature selection and ML
features = FeatureSelector(config).select_features(modules, prepro
model = ModelTrainer(config).train_model(features, labels=sample_g
# Pathway analysis
pathway_res = PathwayAnalyzer(config).analyze_modules(modules, pre
```

This is just an illustrative snippet – in practice the classes will hide many details. The API allows advanced users to insert custom steps or integrate parts of this pipeline with other analyses (e.g., a user can load their data, then only use the network construction and module detection parts).

- **Modularity in Execution**: Because each module is separate, users could choose to run only certain parts. For example, one might skip the ML if only interested in network analysis; or run pathway analysis on a given gene list without rebuilding networks. The CLI can expose flags to run subsets of the pipeline (e.g., `--skip-ml`). Similarly, an API user can call only what they need.
- **Error Handling and Help**: The CLI will provide clear help messages (`itsync --help` describing usage). It will validate that required inputs are present (via config or arguments) and give informative errors (e.g., "Tissue X expression file not found" or "No overlapping donors between tissue A and B – cannot compute cross-tissue correlations" to catch issues early). Progress messages are printed to console so the user knows which step is running (with more detailed info in the log file).

The combination of CLI and API, along with a well-documented configuration system, ensures both **accessibility for non-programmers** (just edit a YAML and run the tool) and **flexibility for developers** (use components in custom Python workflows). This dual approach follows best practices in scientific software, similar to how packages like scikit-learn or ScanPy offer CLI hooks but also rich APIs.

Proposed Package Structure and Extensibility

The project will be organized into a clear folder structure reflecting the modules above. This promotes maintainability and makes it easy to extend or modify parts of the pipeline without breaking others. A suggested structure is:

plaintext Copy

```
intertissue_sync/                # Top-level Python package
├── __init__.py
├── data/                        # Data ingestion & preprocessing
│   ├── __init__.py
│   ├── loader.py                # DataLoader class and file readers
│   └── preprocess.py            # Preprocessor class (normalization,
├── network/                     # Network construction
│   ├── __init__.py
│   ├── coexpression.py          # NetworkBuilder (correlation, adjace
│   └── wgcna_refinement.py      # Optional: functions for multi-beta 
├── modules/                     # Module detection
│   ├── __init__.py
│   ├── detection.py             # ModuleDetector, clustering and dyna
│   └── module.py                # Definition of Module class (genes, 
├── analysis/                    # Differential analysis & stats
│   ├── __init__.py
│   ├── differential.py          # DifferentialAnalyzer (MDC calculati
│   └── enrichment.py            # Enrichment analysis (could also liv
├── ml/                          # Machine learning & feature selectio
│   ├── __init__.py
│   ├── features.py              # FeatureSelector (LASSO and feature 
│   ├── models.py                # ModelTrainer (handles RF, XGB, CV, 
│   └── metrics.py               # Metric computations (ROC, AUC, etc.
├── pathways/                    # Pathway analysis
│   ├── __init__.py
│   ├── gene_sets.py             # Functions to load/manage gene sets
│   ├── ssgsea.py                # ssGSEA computation functions
```

```
│   └── pathway_analysis.py      # PathwayAnalyzer (combining enrichme
├── visualization/               # Visualization routines
│   ├── __init__.py
│   └── plots.py                 # Plot functions (heatmaps, dendrogra
├── integration/                 # R integration via rpy2
│   ├── __init__.py
│   ├── r_wgcna.py               # Interfaces to R for WGCNA (TOM, dyn
│   └── r_clusterprofiler.py     # Interfaces to R for GO/KEGG enrichm
├── cli.py                       # CLI definitions (using argparse or
└── utils/                       # (Optional) common utilities, e.g.,
    ├── __init__.py
    ├── config.py                # ConfigManager to parse YAML/JSON
    └── logger.py                # Logger setup for the package
```

Additionally, top-level files like `setup.py` or `pyproject.toml` will be present for packaging, and a `README.md` with usage examples. A `tests/` directory will contain unit and integration tests for each module to ensure reliability (important for open-source readiness).

Extensibility: This modular structure allows new functionality to be added with minimal friction:

- To support a new **data type** or format, one can add a new function in `loader.py` or extend `Preprocessor` for specialized normalization (for example, adding a function to handle single-cell RNA-seq data differently).
- To target a different **phenotype** (say, disease vs control rather than aging), no code changes are required – the user would supply a different sample grouping in the config. The MDC calculation and ML classification naturally apply to any binary (or multi-class with some extension) outcome.
- Incorporating another **omics layer** (like proteomics) in the network could be done by treating it as another "tissue" layer in the coexpression network or by writing a new network construction module that merges correlation across data types. The rest of the pipeline (module detection, MDC, etc.) operates on correlation networks generally, so it would remain applicable.

- For **cross-species** analysis, one might include a step to map orthologous genes between species and then feed the combined data into the pipeline. The design doesn't hard-code any species-specific assumptions (aside from perhaps gene naming conventions in enrichment which can be configured), so it could be used for mouse or other organisms' multi-tissue data by providing the appropriate gene sets and annotations.

Finally, the package will follow best practices for open-source computational biology software: clear documentation (docstrings and a usage guide), version control, and continuous integration testing to ensure new changes don't break existing features. By building on standard libraries (pandas, numpy, sklearn, etc.) and well-tested R packages for critical steps, the tool will be robust and trustworthy for researchers. **Modularity and clarity** are prioritized so that users can understand each step of the pipeline or modify it to explore new research questions in inter-tissue coordination.

References:

- Briller *et al.*, 2025 – Proposed the multi-tissue WGCNA and MDC framework this package is based on nature.com nature.com nature.com .
- Zhang & Horvath, 2005 – WGCNA method for co-expression network analysis nature.com .
- Langfelder & Horvath, 2008 – Dynamic tree cut for module detection (used via WGCNA).
- Hänzelmann *et al.*, 2013 – GSVA (ssGSEA) method for single-sample enrichment scoring nature.com .
- Yu *et al.*, 2012 – clusterProfiler R package for enrichment analysis nature.com .

(Additional references and documentation will be provided within the package manual.)

Citations

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...

https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co…
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co…
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co…
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co…
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co…
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co…
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co…
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co…
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co…
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co…
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co…
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co…
https://www.nature.com/articles/s41598-025-94043-9?error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?
error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?
error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?
error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?
error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?
error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?
error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?
error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?
error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

A computational framework for detecting inter-tissue gene-expression co...
https://www.nature.com/articles/s41598-025-94043-9?
error=cookies_not_supported&code=b3a9fe73-44a7-4880-b8cc-21b394f85d06

All Sources

nature