python_code

August 15, 2025

1 Data pre-processing 2

Load needed libraries:

[ ]:
```
import liana as li
from liana.method import singlecellsignalr, connectome, cellphonedb, natmi,␣
↪logfc, cellchat, geometric_mean
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import gridspec
from mpl_toolkits.axes_grid1 import make_axes_locatable
import scanpy as sc
pd.options.mode.chained_assignment = None  # default='warn'
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
```

Load data:

[169]:
```
adata = sc.read_h5ad(filename='/Users/sabrina/Library/CloudStorage/
↪OneDrive-UniversityofCopenhagen/Thesis/CCC inference/Liana/rna_data.h5ad')
adata.raw = adata
```

Filter data, keeping only genes present in the NeuronChat database:

[ ]:
```
def filter_genes(adata):
    with open('NeuronChat_genes.txt', 'r') as file:
        lines = [line.strip() for line in file]

    NC_genes = lines

    mask = adata.var_names.isin(NC_genes)

    adata_filt = adata[:, mask].copy()
    return adata_filt
adata_filt = filter_genes(adata)
```

```
print(len(adata_filt.var_names)) # there are 22 genes that are in the␣
 ↪NeuronChat database/resource but not in our data.
# After running rank_aggregate, we end up with 52 features.
print(len(adata.var_names))
```

```
178
19377
```

2 Inference of cell-cell communication

Run all methods included in Liana+, as well as the consensus rank agreggation, for both the original and the filtered datasets:

[171]:
```
methods = [cellphonedb, singlecellsignalr, connectome, natmi, cellchat,␣
 ↪geometric_mean]
new_rank_aggregate = li.mt.AggregateClass(li.mt.aggregate_meta, methods=methods)
```

[172]:
```
def run_liana_methods(data, groupby='cell_type', resource_name='mouseconsensu',␣
 ↪use_raw=False, layer='data_SCT', expr_prop=0.1, verbose=True):
    cellphonedb(
        data,
        groupby=groupby,
        resource_name=resource_name,
        use_raw=use_raw,
        layer=layer,
        expr_prop=expr_prop,
        verbose=verbose, key_added='cpdb_res'
    )

    singlecellsignalr(
        data,
        groupby=groupby,
        resource_name=resource_name,
        use_raw=use_raw,
        layer=layer,
        expr_prop=expr_prop,
        verbose=verbose, key_added='scs_res'
    )

    connectome(
        data,
        groupby=groupby,
        resource_name=resource_name,
        use_raw=use_raw,
        layer=layer,
        expr_prop=expr_prop,
        verbose=verbose, key_added='con_res'
```

```
    )

    natmi(
        data,
        groupby=groupby,
        resource_name=resource_name,
        use_raw=use_raw,
        layer=layer,
        expr_prop=expr_prop,
        verbose=verbose, key_added='natmi_res'
    )

    cellchat(
        data,
        groupby=groupby,
        resource_name=resource_name,
        use_raw=use_raw,
        layer=layer,
        expr_prop=expr_prop,
        verbose=verbose, key_added='cc_res'
    )

    geometric_mean(
        data,
        groupby=groupby,
        resource_name=resource_name,
        use_raw=use_raw,
        layer=layer,
        expr_prop=expr_prop,
        verbose=verbose, key_added='gm_res'
    )

    new_rank_aggregate(
        data,
        groupby=groupby,
        resource_name=resource_name,
        use_raw=use_raw,
        layer=layer,
        expr_prop=expr_prop,
        verbose=True, key_added='liana_res'
    )

run_liana_methods(adata, groupby='cell_type', resource_name='mouseconsensus',␣
 ↪use_raw=False, layer='data_SCT', expr_prop=0.1, verbose=True)
run_liana_methods(adata_filt, groupby='cell_type',␣
 ↪resource_name='mouseconsensus', use_raw=False, layer='data_SCT', expr_prop=0.
 ↪1, verbose=True)
```

```
Using resource `mouseconsensus`.
Using the `data_SCT` layer!
Converting to sparse csr matrix!
3267 features of mat are empty, they will be removed.
0.35 of entities in the resource are missing from the data.

Generating ligand-receptor stats for 3126 samples and 912 features

100%|      | 1000/1000 [00:02<00:00, 439.89it/s]
Using resource `mouseconsensus`.
Using the `data_SCT` layer!
Converting to sparse csr matrix!
3267 features of mat are empty, they will be removed.
0.35 of entities in the resource are missing from the data.

Generating ligand-receptor stats for 3126 samples and 912 features

Using resource `mouseconsensus`.
Using the `data_SCT` layer!
Converting to sparse csr matrix!
3267 features of mat are empty, they will be removed.
0.35 of entities in the resource are missing from the data.

Generating ligand-receptor stats for 3126 samples and 912 features

Using resource `mouseconsensus`.
Using the `data_SCT` layer!
Converting to sparse csr matrix!
3267 features of mat are empty, they will be removed.
0.35 of entities in the resource are missing from the data.

Generating ligand-receptor stats for 3126 samples and 912 features

Using resource `mouseconsensus`.
Using the `data_SCT` layer!
Converting to sparse csr matrix!
3267 features of mat are empty, they will be removed.
0.35 of entities in the resource are missing from the data.

Generating ligand-receptor stats for 3126 samples and 912 features

100%|      | 1000/1000 [00:28<00:00, 35.05it/s]
Using resource `mouseconsensus`.
Using the `data_SCT` layer!
Converting to sparse csr matrix!
3267 features of mat are empty, they will be removed.
0.35 of entities in the resource are missing from the data.

Generating ligand-receptor stats for 3126 samples and 912 features

100%|      | 1000/1000 [00:01<00:00, 502.20it/s]
Using resource `mouseconsensus`.
Using the `data_SCT` layer!
Converting to sparse csr matrix!
```

```
3267 features of mat are empty, they will be removed.
0.35 of entities in the resource are missing from the data.

Generating ligand-receptor stats for 3126 samples and 912 features
Running CellPhoneDB

100%|      | 1000/1000 [00:02<00:00, 469.95it/s]

Running SingleCellSignalR
Running Connectome
Running NATMI
Running CellChat

100%|      | 1000/1000 [00:27<00:00, 36.26it/s]

Running Geometric Mean

100%|      | 1000/1000 [00:02<00:00, 485.24it/s]
Using resource `mouseconsensus`.
Using the `data_SCT` layer!
Converting to sparse csr matrix!
19 features of mat are empty, they will be removed.
0.95 of entities in the resource are missing from the data.

Generating ligand-receptor stats for 3126 samples and 52 features

100%|      | 1000/1000 [00:01<00:00, 928.29it/s]
Using resource `mouseconsensus`.
Using the `data_SCT` layer!
Converting to sparse csr matrix!
19 features of mat are empty, they will be removed.
0.95 of entities in the resource are missing from the data.
Using resource `mouseconsensus`.

Generating ligand-receptor stats for 3126 samples and 52 features

Using the `data_SCT` layer!
Converting to sparse csr matrix!
19 features of mat are empty, they will be removed.
0.95 of entities in the resource are missing from the data.

Generating ligand-receptor stats for 3126 samples and 52 features

Using resource `mouseconsensus`.
Using the `data_SCT` layer!
Converting to sparse csr matrix!
19 features of mat are empty, they will be removed.
0.95 of entities in the resource are missing from the data.
Using resource `mouseconsensus`.

Generating ligand-receptor stats for 3126 samples and 52 features

Using the `data_SCT` layer!
Converting to sparse csr matrix!
```

```
19 features of mat are empty, they will be removed.
0.95 of entities in the resource are missing from the data.

Generating ligand-receptor stats for 3126 samples and 52 features

100%|      | 1000/1000 [00:02<00:00, 437.43it/s]
Using resource `mouseconsensus`.
Using the `data_SCT` layer!
Converting to sparse csr matrix!
19 features of mat are empty, they will be removed.
0.95 of entities in the resource are missing from the data.

Generating ligand-receptor stats for 3126 samples and 52 features

100%|      | 1000/1000 [00:01<00:00, 898.63it/s]
Using resource `mouseconsensus`.
Using the `data_SCT` layer!
Converting to sparse csr matrix!
19 features of mat are empty, they will be removed.
0.95 of entities in the resource are missing from the data.

Generating ligand-receptor stats for 3126 samples and 52 features
Running CellPhoneDB

100%|      | 1000/1000 [00:01<00:00, 991.28it/s]

Running SingleCellSignalR
Running Connectome
Running NATMI
Running CellChat

100%|      | 1000/1000 [00:02<00:00, 432.99it/s]

Running Geometric Mean

100%|      | 1000/1000 [00:01<00:00, 988.57it/s]
```

[173]:
```
method_keys = {
    'CellPhoneDB' : ('cpdb_res', 'lr_means', 'cellphone_pvals'),
    'SingleCellSignalR' : ('scs_res', 'lrscore', None),
    'Connectome' : ('con_res', 'expr_prod', 'scaled_weight'),
    'NATMI' : ('natmi_res', 'expr_prod', 'spec_weight'),
    'CellChat' : ('cc_res', 'lr_probs', 'cellchat_pvals'),
    'Geometric Mean' : ('gm_res', 'lr_gmeans', 'gmean_pvals'),
    'Liana+ Consensus' : ('liana_res', 'magnitude_rank', 'specificity_rank')
}
```

[174]:
```
adata.uns['cc_res']
```

[174]:

	ligand	ligand_complex	ligand_props	ligand_trimean	mat_max	receptor	\
8417	Nxph1	Nxph1	1.000000	0.435470	6.828712	Nrxn1	
13266	Nxph1	Nxph1	1.000000	0.435470	6.828712	Nrxn1	

```
24284   Nxph1          Nxph1      1.000000        0.435470  6.828712    Nrxn1
15618   Nxph1          Nxph1      1.000000        0.435470  6.828712    Nrxn1
26719   Nxph1          Nxph1      1.000000        0.435470  6.828712    Nrxn1
…       …              …          …               …         …           …
15045    Calr           Calr      0.142857        0.000000  6.828712     Lrp1
5042   Col4a1         Col4a1      0.130435        0.000000  6.828712    Itga3
5041     Agrn           Agrn      0.445652        0.025376  6.828712     Dag1
15048  Sema3a         Sema3a      0.136364        0.000000  6.828712     Nrp1
0      Adam10         Adam10      0.344000        0.025376  6.828712  Tspan12

      receptor_complex  receptor_props  receptor_trimean    source     target  \
8417              Nrxn1        0.993506          0.411159  V1-Rensh        V0v
13266             Nrxn1        1.000000          0.407146  V1-Rensh  V1-Pou6f2
24284             Nrxn1        1.000000          0.405443  V1-Rensh        V2b
15618             Nrxn1        1.000000          0.405387  V1-Rensh   V1-Rensh
26719             Nrxn1        1.000000          0.401294  V1-Rensh         V3
…                 …             …              …         …           …
15045              Lrp1        0.352941          0.025376       V0v   V1-Rensh
5042       Itga3_Itgb1        0.230769          0.000000        V3         MN
5041              Dag1        0.233846          0.000000        V3         MN
15048      Nrp1_Plxna2        0.147059          0.000000       V0v   V1-Rensh
0              Tspan12        0.132000          0.000000       DI6        DI6

       lr_probs  cellchat_pvals
8417   0.263674             0.0
13266  0.261774             0.0
24284  0.260965             0.0
15618  0.260939             0.0
26719  0.258986             0.0
…         …              …
15045  0.000000             1.0
5042   0.000000             1.0
5041   0.000000             1.0
15048  0.000000             1.0
0      0.000000             1.0

[25739 rows x 13 columns]
```

2.1 Cell type-cell type communication

2.1.1 Frequency of interactions for each CT pair

For each method, false postive filtering is required.

Dimitrov et. al (2024):

"Then we ran all ligand–receptor methods in LIANA+ without taking spatial information into account. AUROC was calculated for the whole distribution of each ligand–receptor method's scoring functions. To calculate balanced accuracy and normalized F1 (below), **we used false positive**

filtering thresholds as suggested by each of the methods' authors (if available). - For CellPhoneDB, CellChat and Geometric mean, interactions with P values below 0.05 were filtered. - For CellChat's ligand–receptor (LR) probabilities, we additionally only kept interactions for which either the ligand or receptor P values were under 0.05. - Similarly, for Connectome and log2 fold changes (log2FC), interactions were kept only if both ligand and receptor P values were under 0.05 and had a positive scaled weight or log-fold change, respectively. - For SingleCellSignalR, we considered ligand–receptor interactions with LR scores above 0.6. - For LIANA's rank aggregate, we kept only those with a magnitude rank <0.05. - Since the authors of NATMI and scSeqComm do not suggest a threshold, we kept interactions if they were within the top 5% of the specificity weight and interscore distributions, respectively. **Similarly, when evaluating the individual scoring functions from each method, we considered only the top 5% as positive predictions.**"

[175]:
```
def dict_of_matrices(adata, adata_filt, method_keys):
    """
    Create a dictionary of communication matrices for each method.
    """
    freq_matrices = {'specificity': {}, 'magnitude': {}}
    freq_matrices_filt = {'specificity': {}, 'magnitude': {}}
    cell_types = ['DI6',
    'MN',
    'V0d',
    'V0v',
    'V1-Foxp2',
    'V1-Pou6f2',
    'V1-Rensh',
    'V1-Sp8',
    'V2a-1',
    'V2a-2',
    'V2b',
    'V3']
    from scipy.stats import norm
    import scipy
    for method, (key, mag_col, spec_col) in method_keys.items():
        freq_matrices['magnitude'][method] = {}
        freq_matrices_filt['magnitude'][method] = {}

        for data_name, data, matrices_dict in [('adata', adata, freq_matrices),␣
↪('adata_filt', adata_filt, freq_matrices_filt)]:
            df = data.uns[key].copy()
            if method != 'Liana+ Consensus':
                threshold = df[mag_col].quantile(0.95)
                df = df[df[mag_col] >= threshold]
                if method in ['CellPhoneDB', 'CellChat', 'Geometric Mean']:
                    df = df[df[spec_col] < 0.05]
                elif method == 'SingleCellSignalR':
                    df = df[df[mag_col] > 0.6]
                elif method == 'Connectome':
```

```
                    df = df[df[spec_col] >= 0]
                elif method == 'NATMI':
                    threshold = df[spec_col].quantile(0.95)
                    df = df[df[spec_col] >= threshold]
            else:
                df = df[df[mag_col] < 0.05]

            matrices_dict['magnitude'][method]["Frequency"] = df.
 ↪assign(score=df[mag_col]).groupby(['source', 'target'])['score'].size().
 ↪unstack(fill_value=0).reindex(index=cell_types, columns=cell_types,␣
 ↪fill_value=0).T
            matrices_dict['magnitude'][method]["Weight"] = df.
 ↪assign(score=df[mag_col]).groupby(['source', 'target'])['score'].sum().
 ↪unstack(fill_value=0).reindex(index=cell_types, columns=cell_types,␣
 ↪fill_value=0).T
            print(method, len(df)) if data_name == 'adata_filt' else None

    return freq_matrices, freq_matrices_filt
freq_matrices, freq_matrices_filt = dict_of_matrices(adata, adata_filt,␣
 ↪method_keys)
```

```
CellPhoneDB 82
SingleCellSignalR 82
Connectome 82
NATMI 6
CellChat 82
Geometric Mean 82
Liana+ Consensus 255
```

3 Assessment of cell-cell communication inference methods

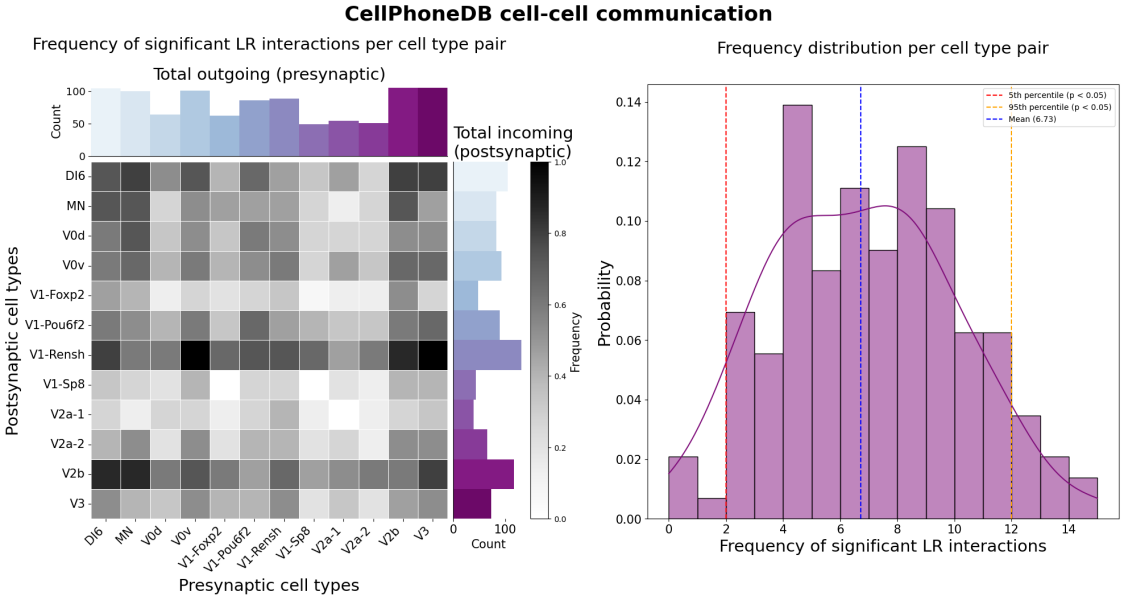
[ ]:
```
def plot_heatmaps(comm_matrix_count, comm_matrix_weight, method_key, data_name,␣
 ↪percentiles=(5, 95)):
    method, key, mag_col, spec_col = method_key

    matrices = [
        (comm_matrix_count, "frequency", "Greys", "Frequency"),
        (comm_matrix_weight, "weighted sum", "Greys", "Sum of scores")
    ]

    for i, (comm_matrix, matrix_label, cmap, cbar_label) in enumerate(matrices):
            fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))
            gs = gridspec.GridSpec(
            2, 4,
            width_ratios=[1, 0.26, 0.04, 1],
            height_ratios=[0.25, 1],
            wspace=0.05, hspace=0.05)
```

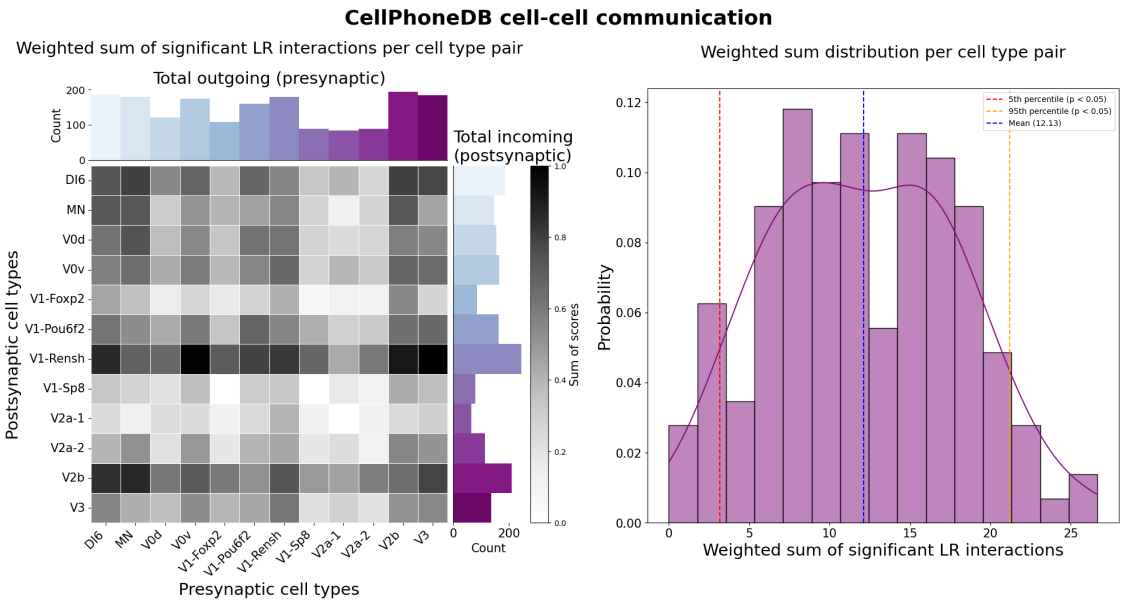
```
            scores = comm_matrix.values.flatten()
            thresholds = np.percentile(scores, percentiles)

            divider = make_axes_locatable(ax1)
            ax_top = divider.append_axes("top", size="20%", pad=0.1, sharex=ax1)
            ax_right = divider.append_axes("right", size="20%", pad=0.1,␣
↪sharey=ax1)
            ax_cbar = divider.append_axes("right", size="5%", pad=0.1)

            cell_types = comm_matrix.index.to_list()
            row_sums = comm_matrix.loc[cell_types].sum(axis=1) # target
            col_sums = comm_matrix[cell_types].sum(axis=0) # source

            palette = sns.color_palette("BuPu" if data_name=="Full data" else␣
↪"RdPu", len(cell_types))
            celltype_color_dict = {cell: palette[j] for j, cell in␣
↪enumerate(cell_types)}
            col_colors = [celltype_color_dict[cell] for cell in comm_matrix.
↪columns]
            row_colors = [celltype_color_dict[cell] for cell in comm_matrix.
↪index]

            n = len(cell_types)

            hm = sns.heatmap(
                comm_matrix/comm_matrix.values.max(),
                ax=ax1,
                annot=False,
                fmt='.0f' if matrix_label == "frequency" else '.2f',
                cmap=cmap,
                linewidths=0.6,
                cbar=False,
                square=True)

            ax1.set_xlabel('Presynaptic cell types', fontsize=21)
            ax1.set_ylabel('Postsynaptic cell types', fontsize=21)
            plt.setp(ax1.get_xticklabels(), rotation=45, ha='right',␣
↪fontsize=15)
            plt.setp(ax1.get_yticklabels(), rotation=0, fontsize=15)
            ax1.set_ylim(n, 0)

            ax_top.bar(
                x=np.arange(n) + 0.5,
                height=col_sums.values,
                color=col_colors,
```

```
            width=1.0,
            align='center'
        )
        ax_top.set_xlim(ax1.get_xlim())
        ax_top.set_title('Total outgoing (presynaptic)', fontsize=21)
        ax_top.spines['bottom'].set_visible(True)
        ax_top.spines['right'].set_visible(False)
        ax_top.spines['top'].set_visible(False)
        ax_top.spines['left'].set_visible(False)
        ax_top.tick_params(axis='x', which='both', bottom=False,␣
↪labelbottom=False)
        ax_top.tick_params(axis='y', which='both', left=True, labelsize=13)
        ax_top.set_ylabel('Count', fontsize=15)
        ax_top.yaxis.set_label_position('left')
        ax_top.yaxis.tick_left()

        ax_right.barh(
            y=np.arange(n) + 0.5,
            width=row_sums.values,
            color=row_colors,
            height=1.0,
            align='center'
        )
        ax_right.set_title('Total incoming\n(postsynaptic)', fontsize=21,␣
↪loc='left')
        ax_right.spines['left'].set_visible(True)
        ax_right.spines['top'].set_visible(False)
        ax_right.spines['right'].set_visible(False)
        ax_right.spines['bottom'].set_visible(False)
        ax_right.tick_params(axis='y', which='both', left=False,␣
↪labelleft=False)
        ax_right.tick_params(axis='x', which='both', bottom=True,␣
↪labelsize=15)
        ax_right.set_xlabel('Count', fontsize=15)

        cbar = plt.colorbar(hm.get_children()[0], cax=ax_cbar,␣
↪label=cbar_label)
        cbar.set_label(cbar_label, size=15)



        ax1.set_title(f'{matrix_label.capitalize()} of significant LR␣
↪interactions per cell type pair', fontsize=21, y=1.3)
```

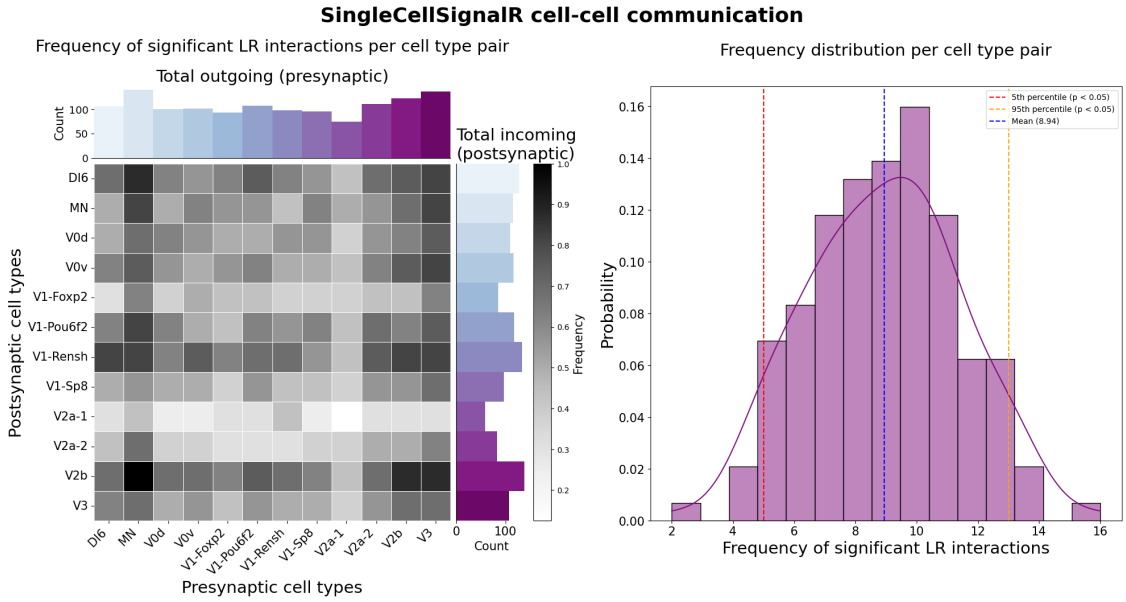
```
            sns.histplot(comm_matrix.values.flatten(), color="#810f7c" if␣
↪data_name=="Full data" else "#f768a1", stat='probability', bins=15 if␣
↪data_name=="Full data" else 3, kde=True, ax=ax2)
            ax2.axvline(x=thresholds[0], color='red', linestyle='--',␣
↪label=f'{percentiles[0]}th percentile (p < 0.05)')
            ax2.axvline(x=thresholds[1], color='orange', linestyle='--',␣
↪label=f'{percentiles[1]}th percentile (p < 0.05)')
            ax2.axvline(x=np.mean(scores), color='blue', linestyle='--',␣
↪label=f'Mean ({np.mean(scores):.2f})')
            ax2.legend()
            ax2.set_title(f"{matrix_label.capitalize()} distribution per cell␣
↪type pair", fontsize=21, y=1.06)
            ax2.set_xlabel(f"{matrix_label.capitalize()} of significant LR␣
↪interactions", fontsize=21)
            ax2.set_ylabel('Probability', fontsize=21)
            plt.setp(ax2.get_xticklabels(), fontsize=15)
            plt.setp(ax2.get_yticklabels(), fontsize=15)
            plt.suptitle(f'{key} cell-cell communication', fontsize=26,␣
↪weight='bold', y=1)
            plt.tight_layout()
            plt.show()
            fig.savefig(f'/Users/sabrina/Library/CloudStorage/
↪OneDrive-UniversityofCopenhagen/Thesis/Figures/
↪F1_{method}_{matrix_label}_{data_name}.svg', format='svg', dpi=300,␣
↪bbox_inches='tight')


    plt.subplots_adjust(top=1.4, bottom=0.08, left=0.10, right=0.95, hspace=0.
↪25,
                        wspace=0.35)
```

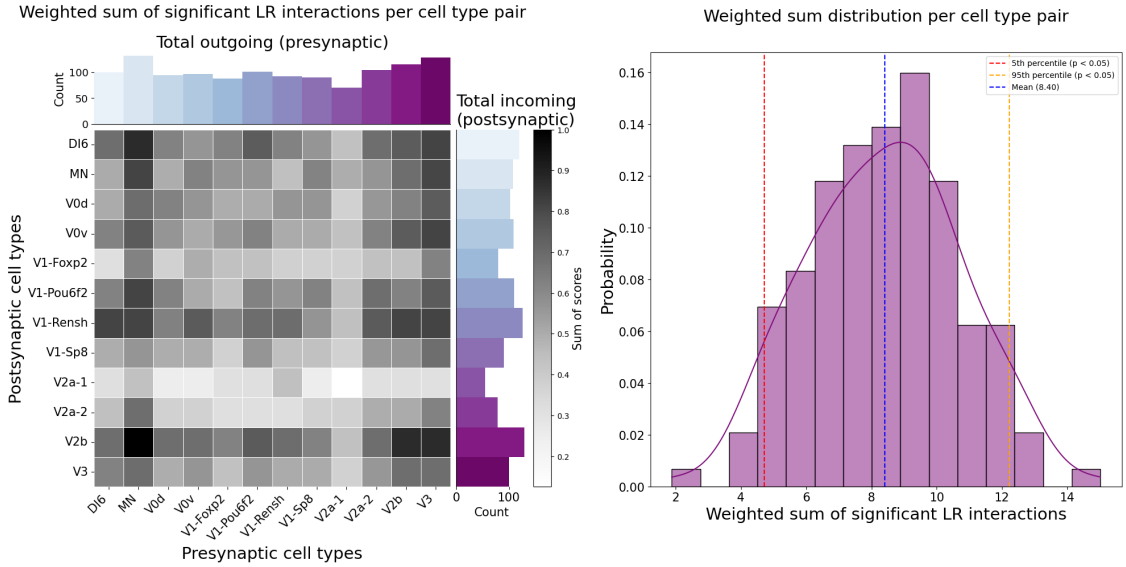
[194]:
```
survey_pairs = pd.read_csv('../survey_pairs.csv', index_col=0).T
for data_name, data, mat_dict in [('Full data', adata, freq_matrices),␣
↪('Filtered data', adata_filt, freq_matrices_filt)]:
    for method, (key, mag_col, spec_col) in method_keys.items():
                freq_mat = mat_dict['magnitude'][method]["Frequency"]
                weight_mat = mat_dict['magnitude'][method]["Weight"]
                plot_new_heatmap(freq_mat, weight_mat, (key, method, mag_col,␣
↪spec_col), data_name)
                #plot_violin(mat_dict['magnitude'][method], (key, method,␣
↪mag_col, spec_col))
                #plot_survey_pairs_violin(freq_mat,(method, key, mag_col,␣
↪spec_col), data_name, survey_pairs=survey_pairs)
```



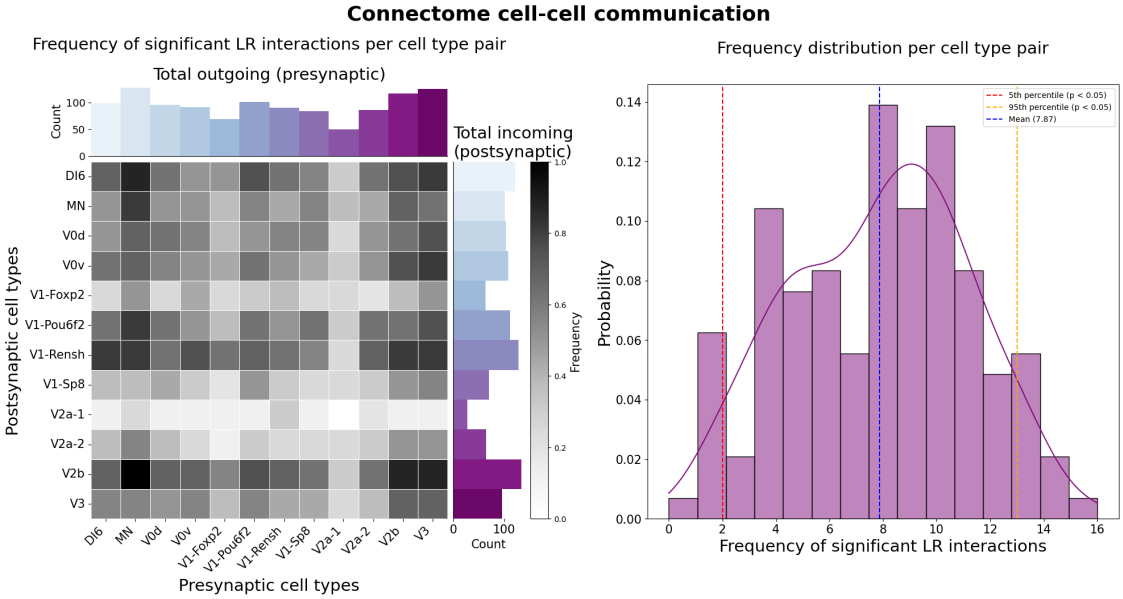


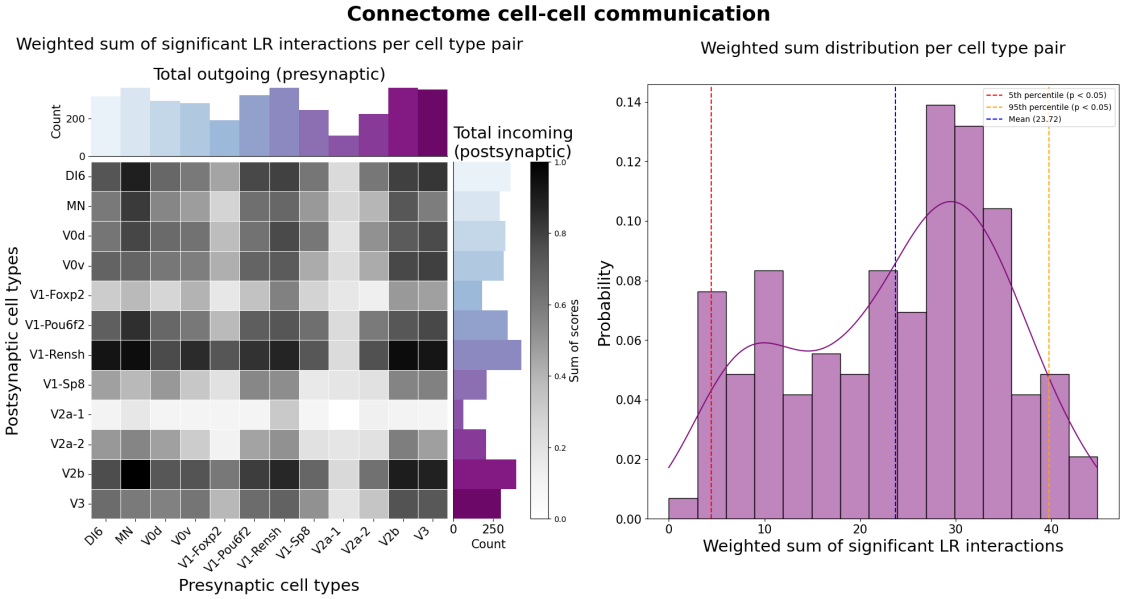
<Figure size 640x480 with 0 Axes>



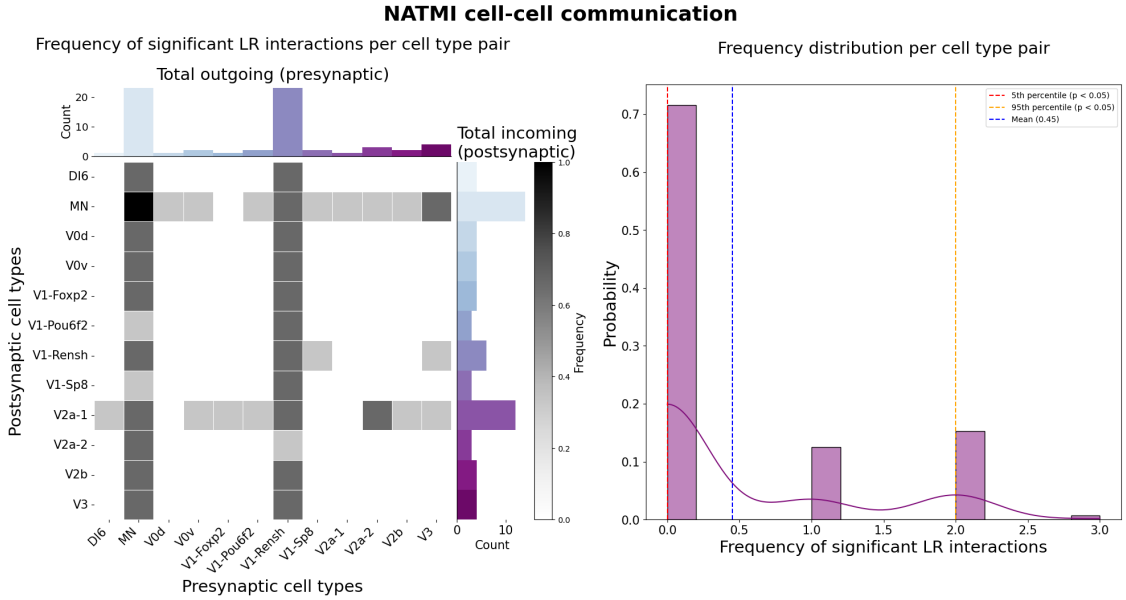


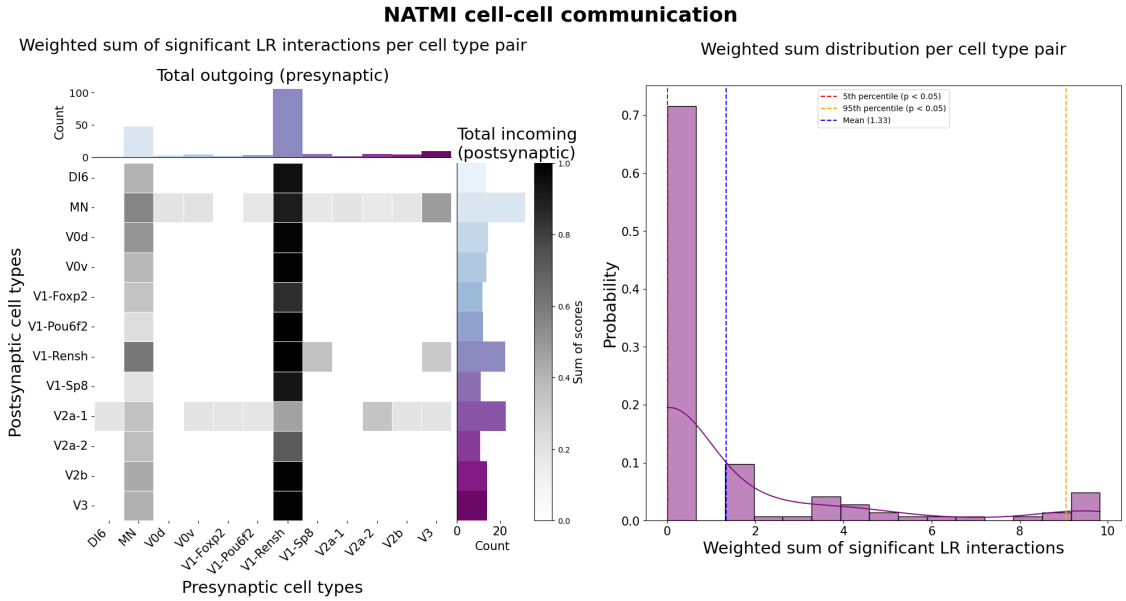
<Figure size 640x480 with 0 Axes>



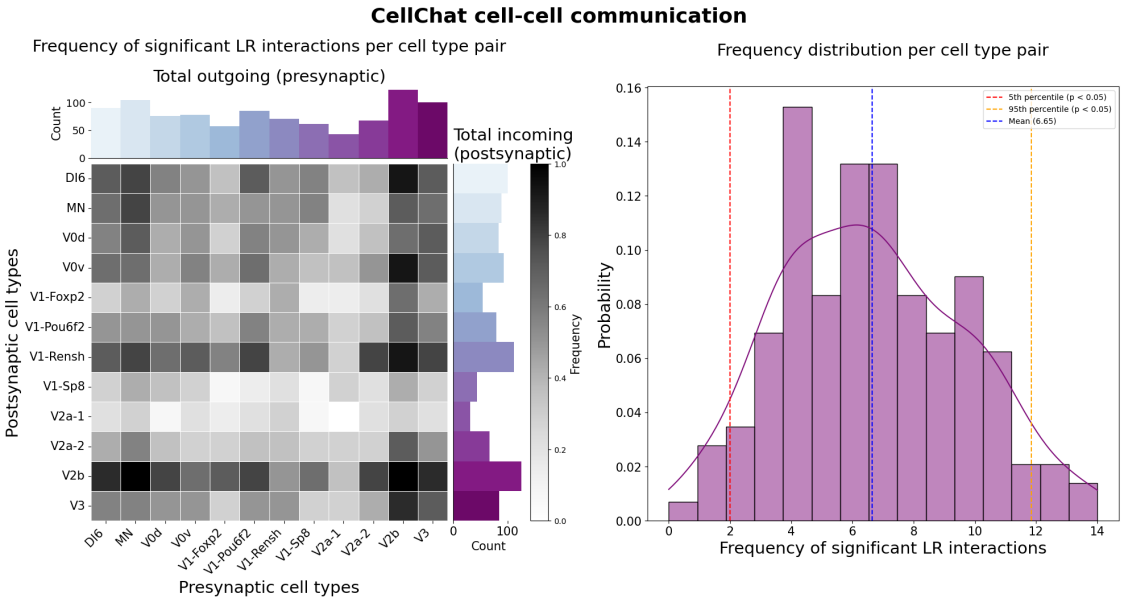


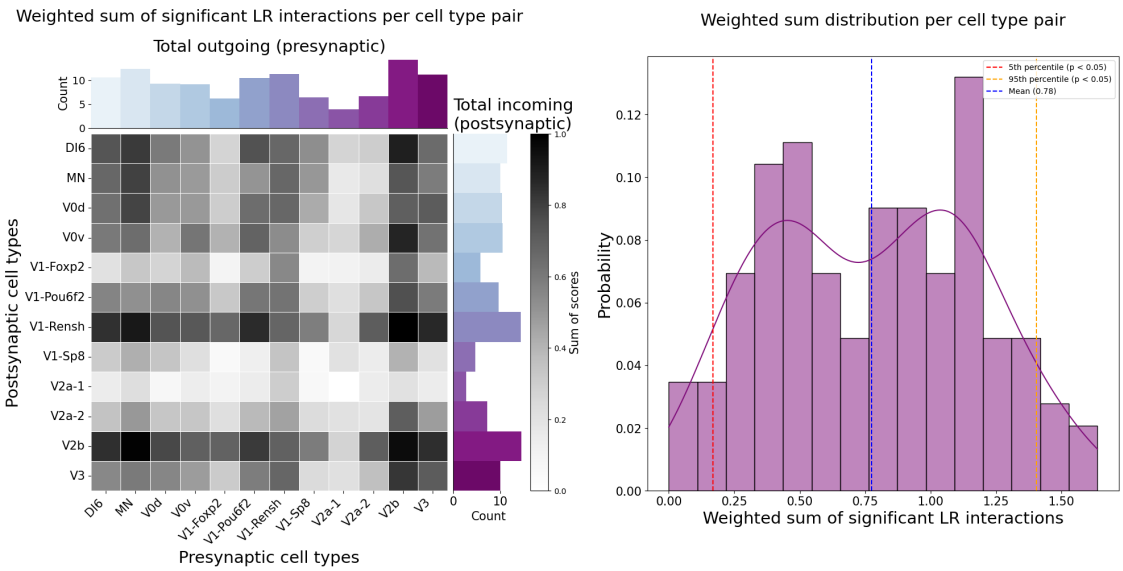
<Figure size 640x480 with 0 Axes>



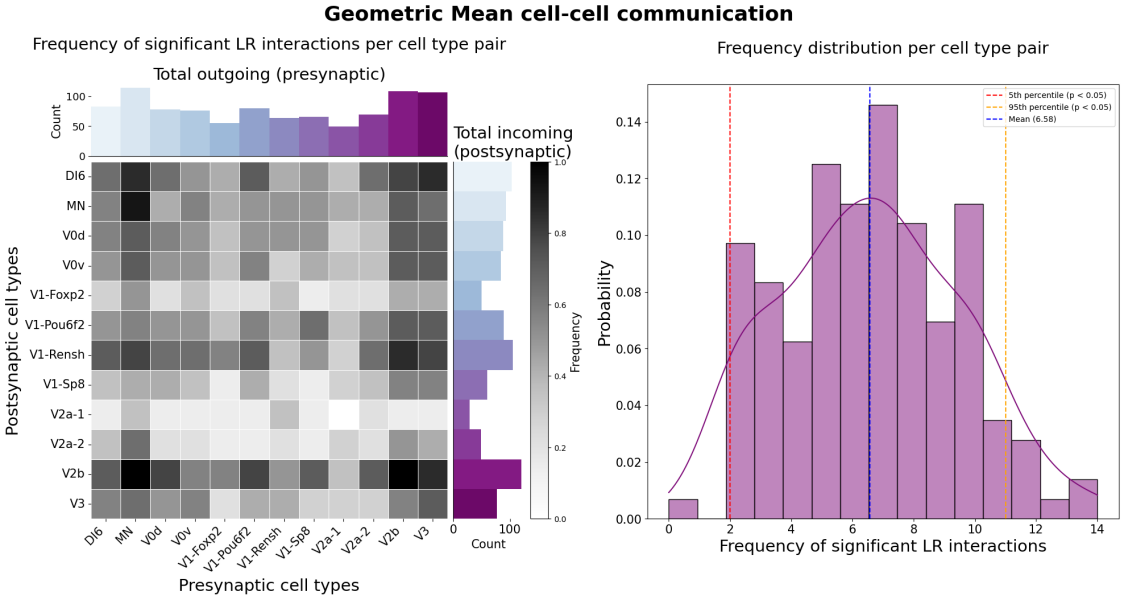


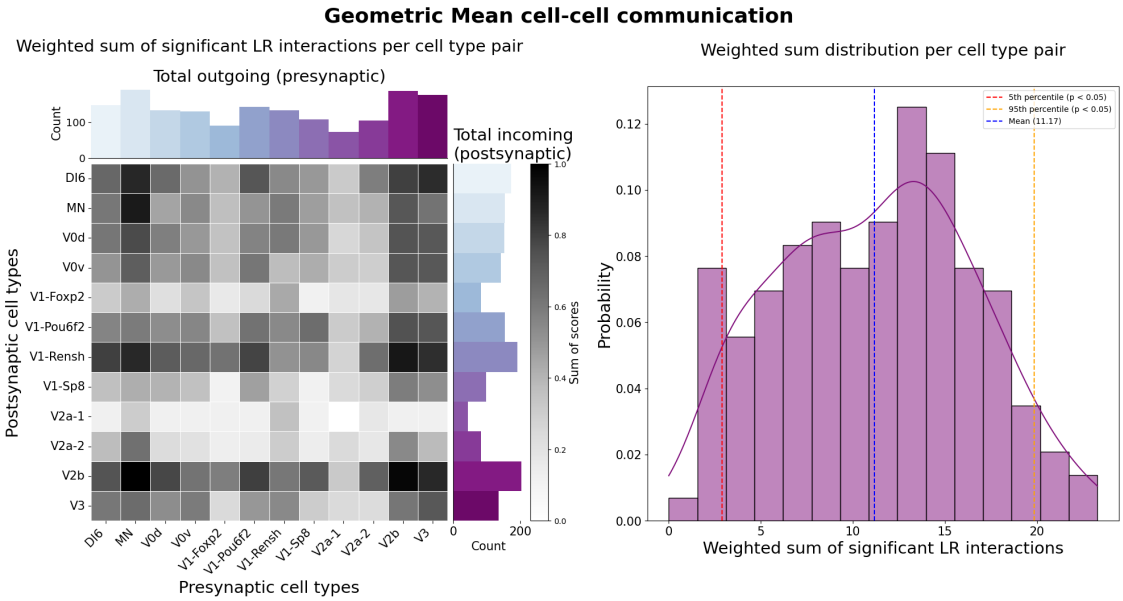
<Figure size 640x480 with 0 Axes>



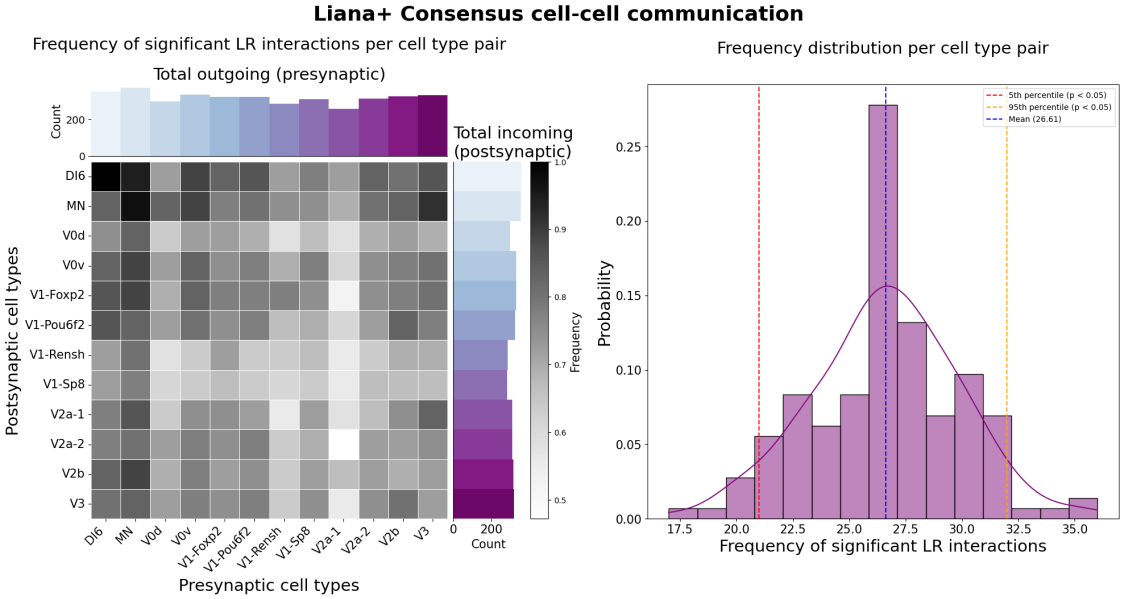


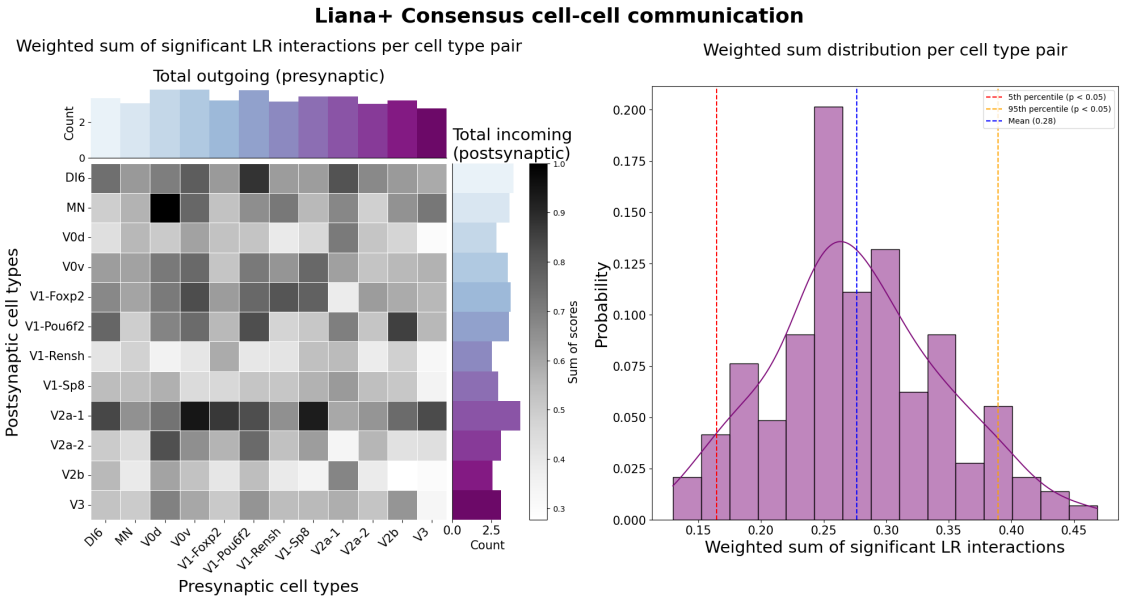
<Figure size 640x480 with 0 Axes>



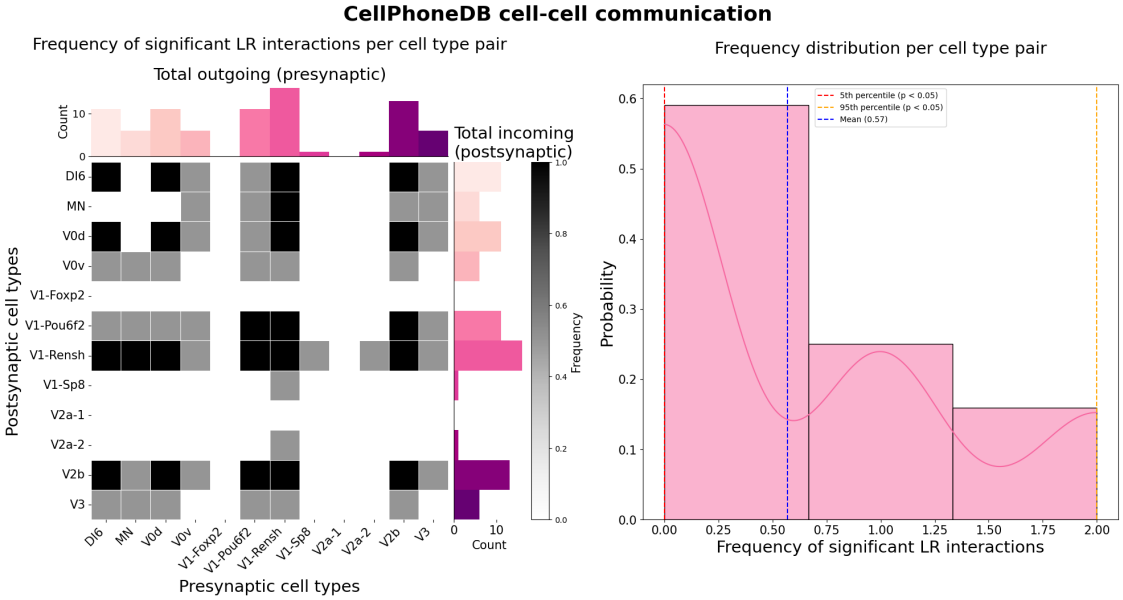


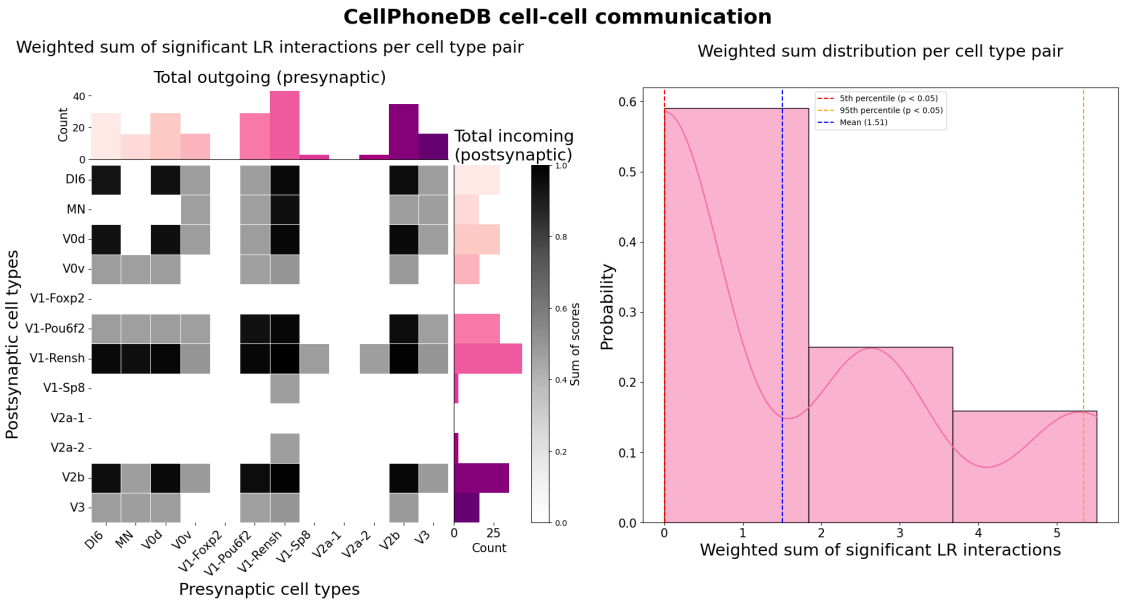
<Figure size 640x480 with 0 Axes>



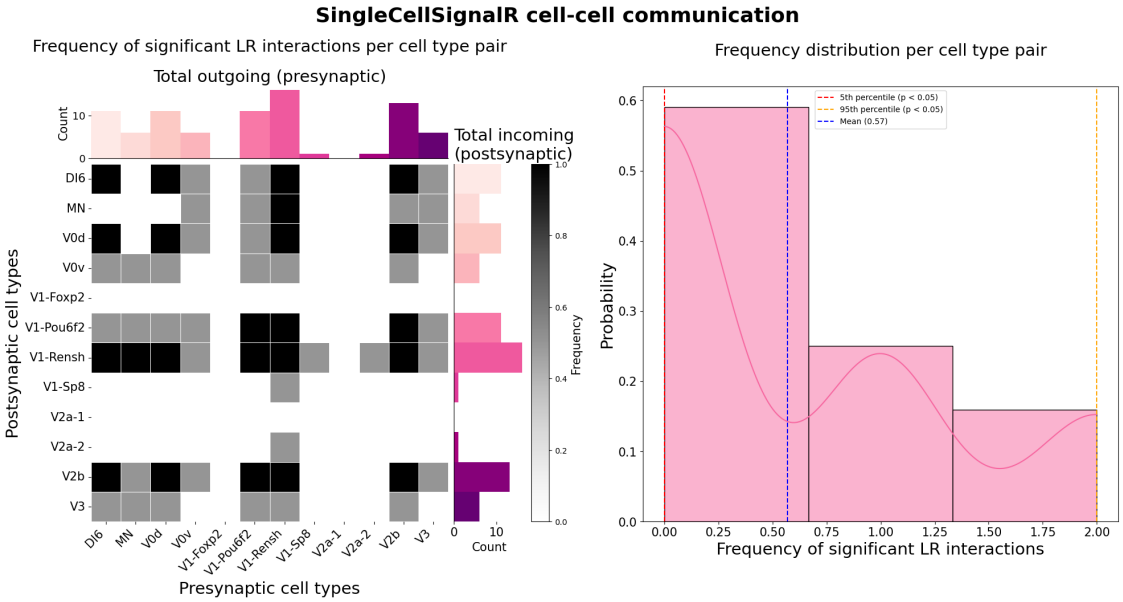


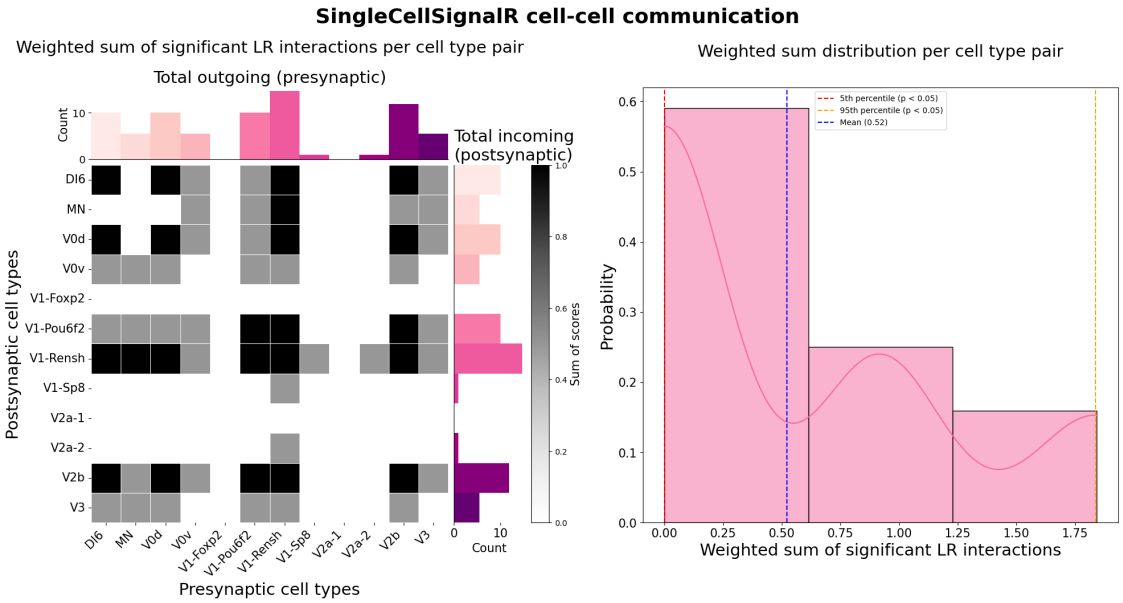
<Figure size 640x480 with 0 Axes>



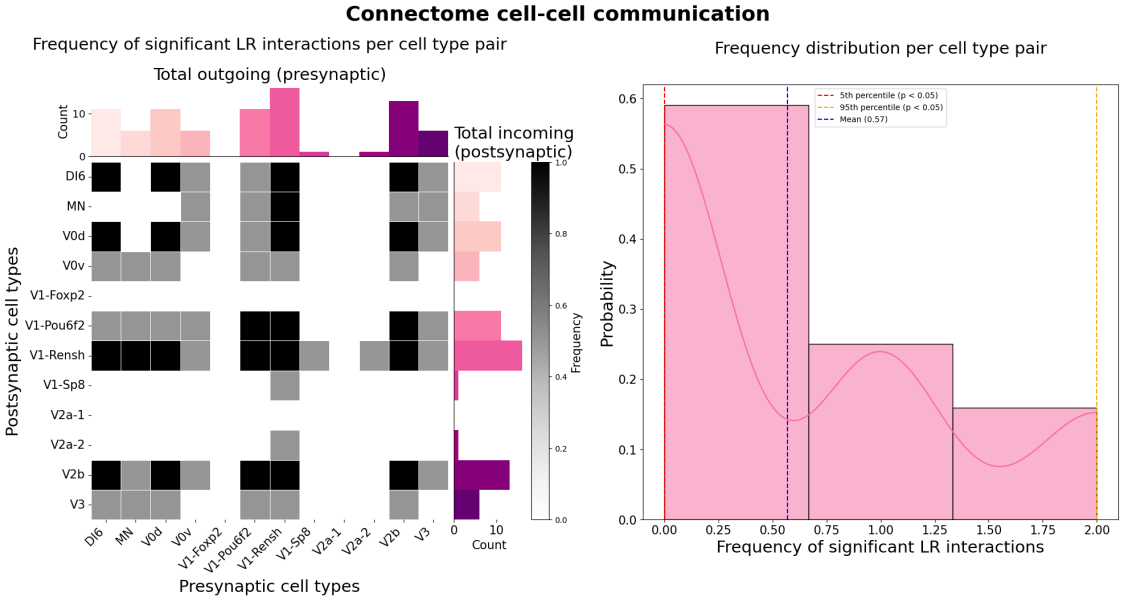


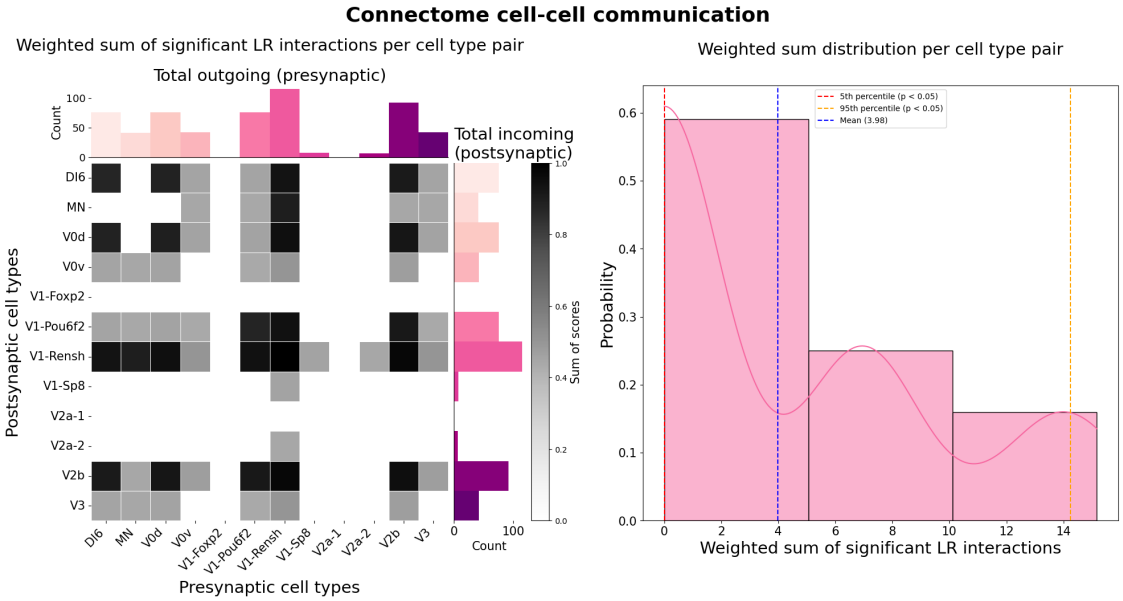
<Figure size 640x480 with 0 Axes>



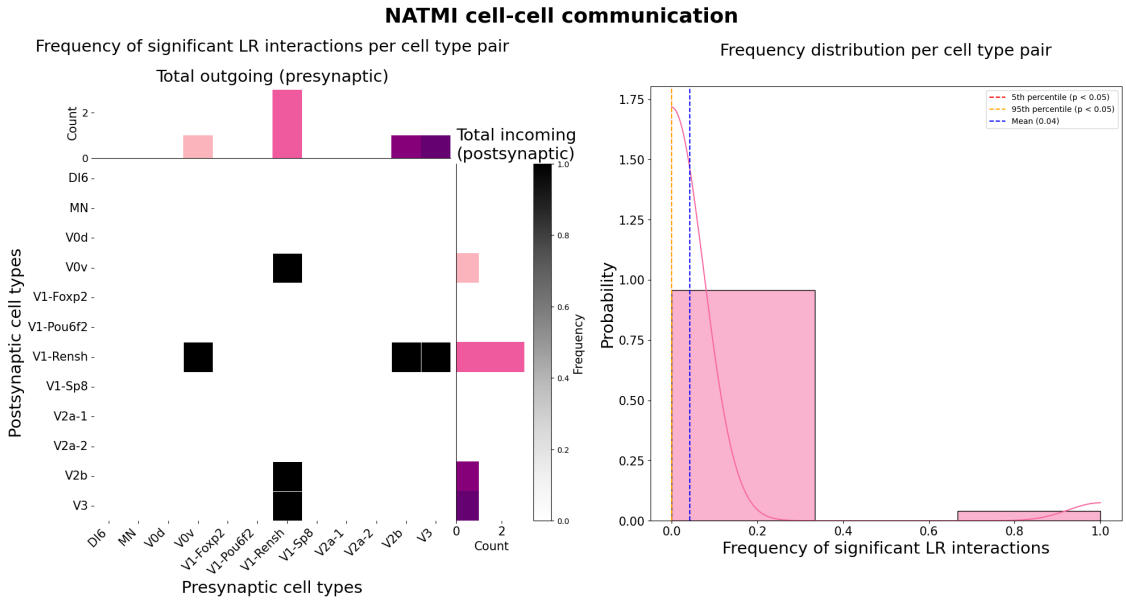


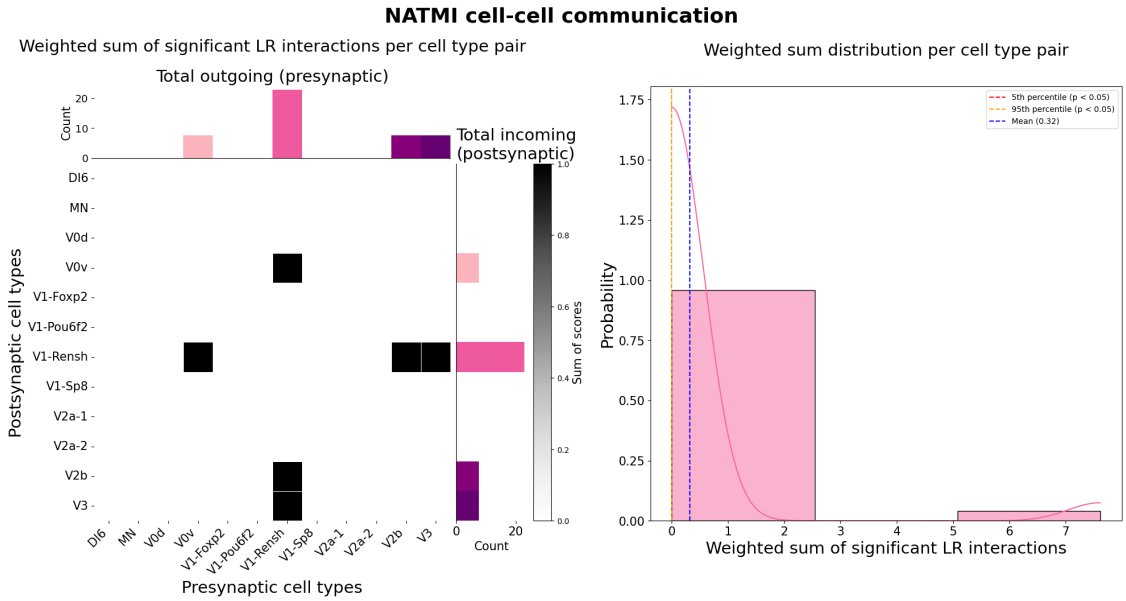
<Figure size 640x480 with 0 Axes>



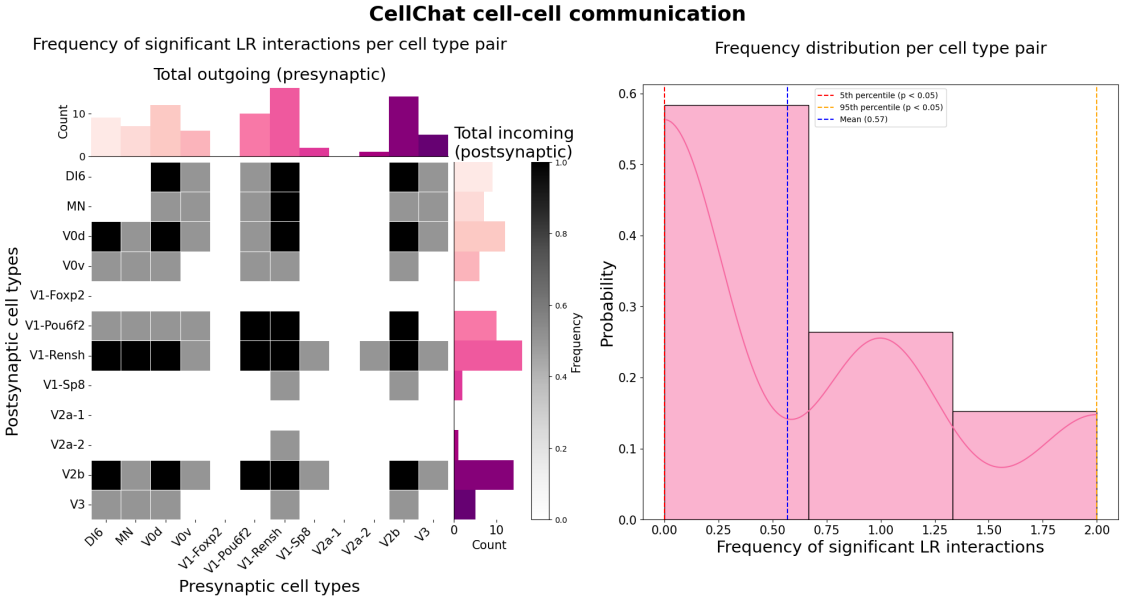


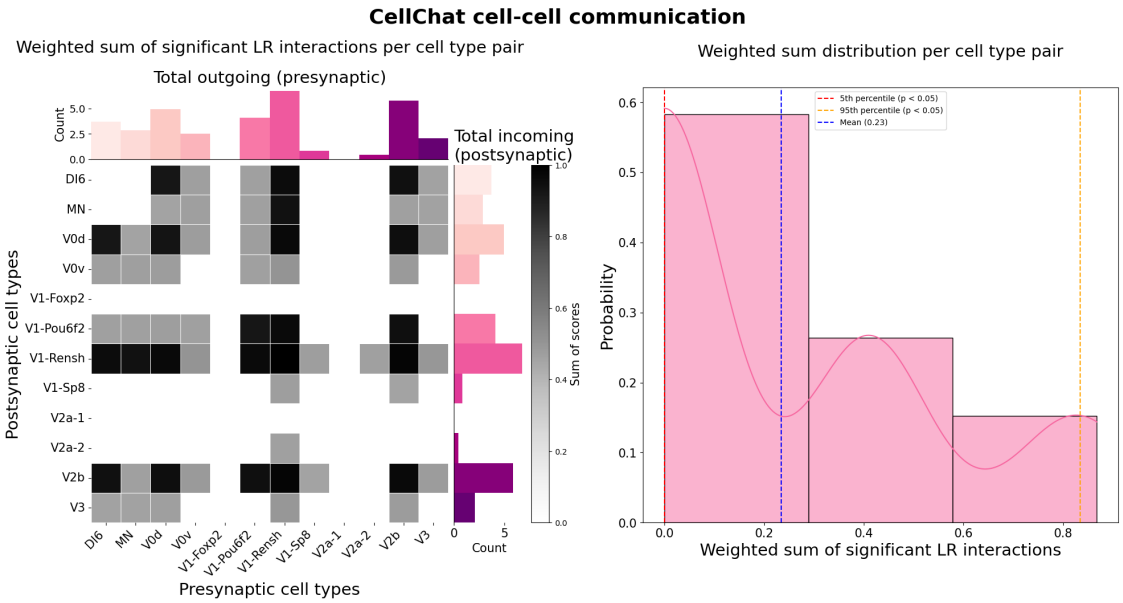
<Figure size 640x480 with 0 Axes>



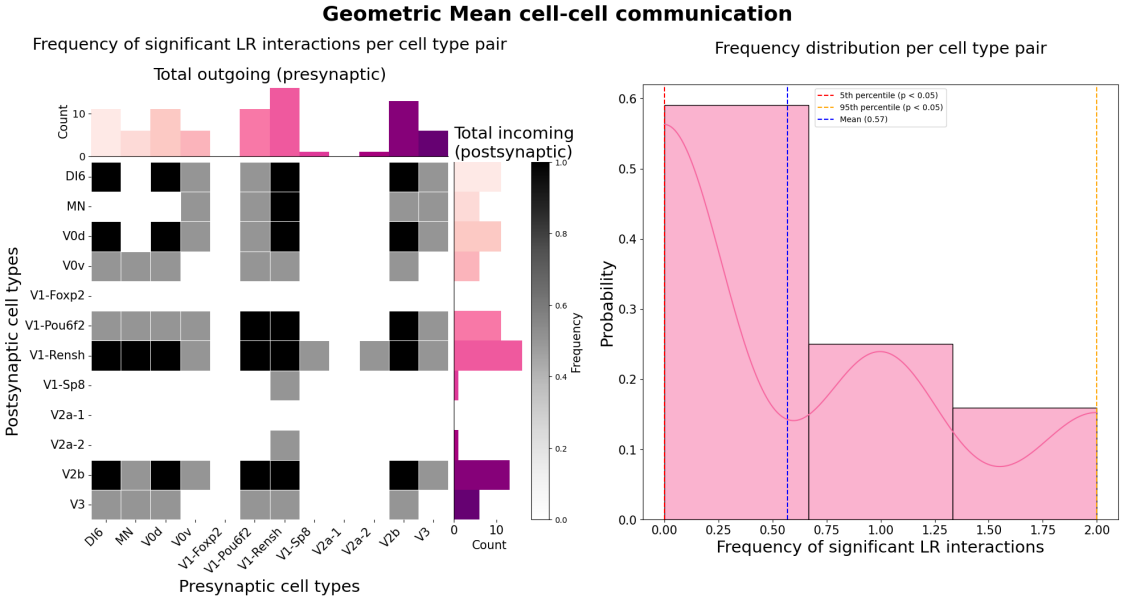


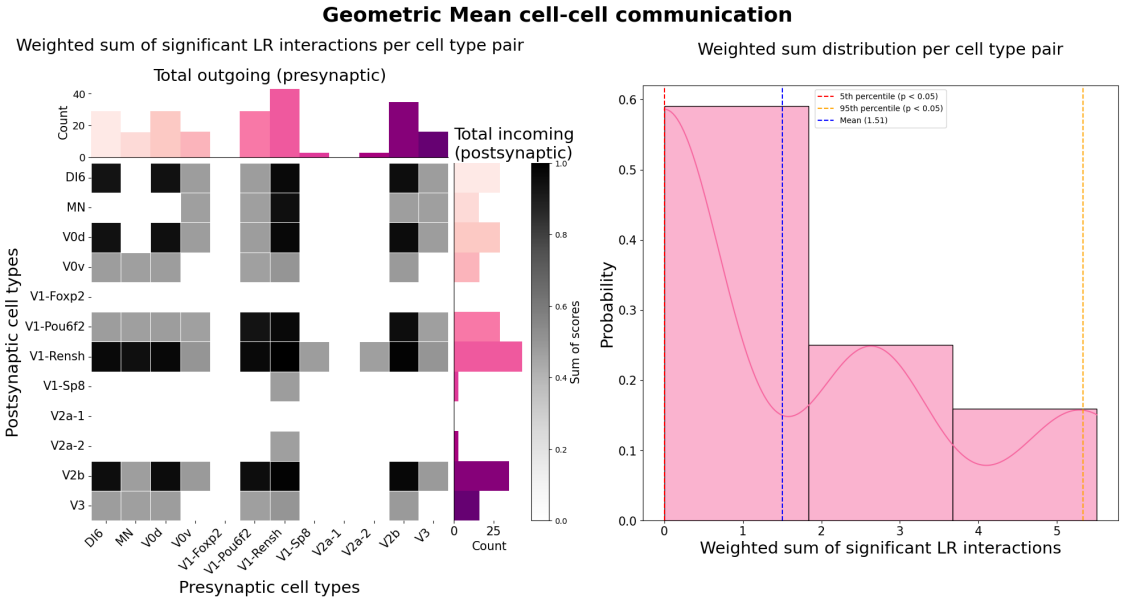
<Figure size 640x480 with 0 Axes>



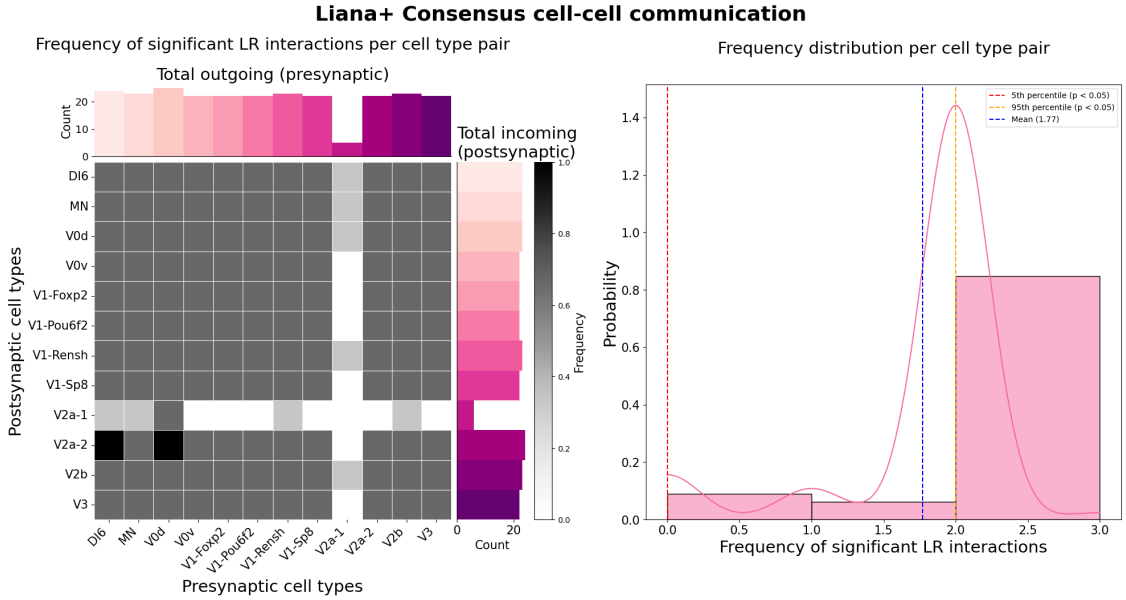


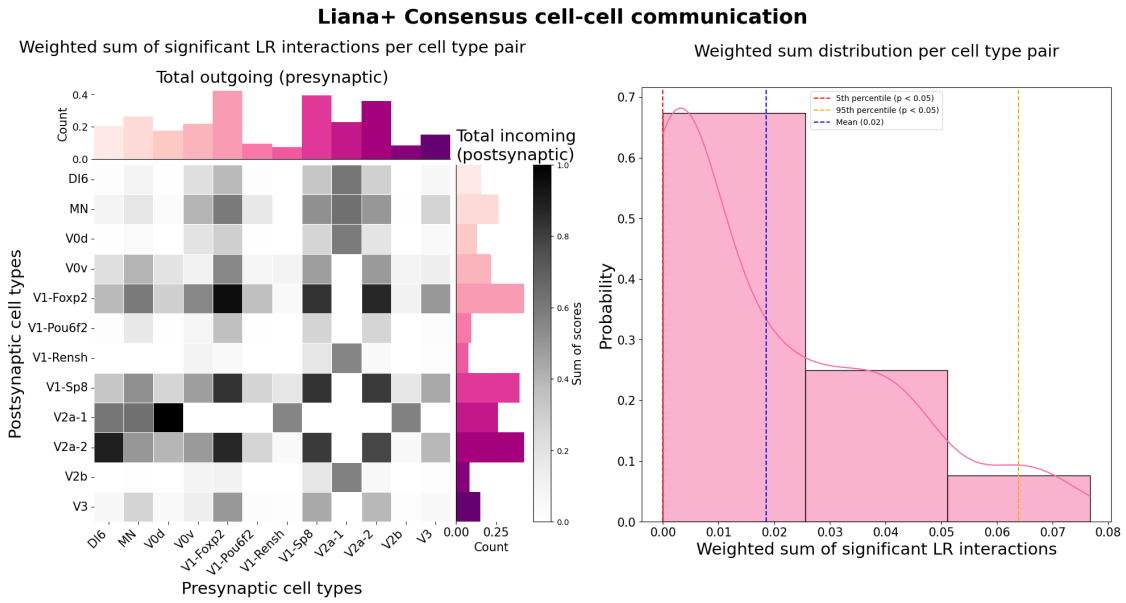
<Figure size 640x480 with 0 Axes>





<Figure size 640x480 with 0 Axes>





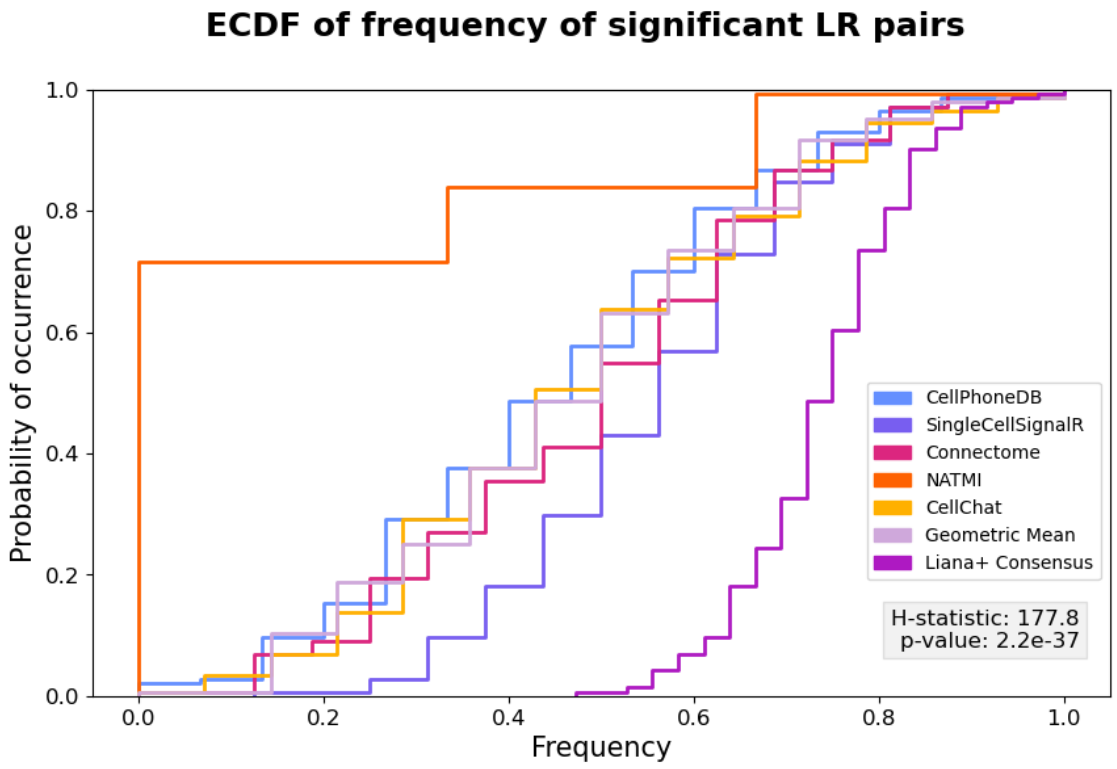
<Figure size 640x480 with 0 Axes>

[ ]:
```
def distribution_comparison(freq_matrices, method_keys, survey_pairs=pd.
↪read_csv('../survey_pairs.csv', index_col=0).T):
    from scipy.stats import kruskal
    import scikit_posthocs as sp
    from matplotlib import patches as mpatches
    from matplotlib import colors as mcolors
```

```
    colors = ["#648FFF", "#785EF0","#DC267F", "#FE6100", "#FFB000", "#CFA8DB",␣
↪"#AE19C2"]

    fig, ax = plt.subplots(1, 1, figsize=(10, 6))
    fig.subplots_adjust(hspace=0.2, wspace=0.2)
    significant_pairs = pd.DataFrame(columns=['Group1', 'Group2', 'p-value'])

    for i, method in enumerate(method_keys.keys()):
        color = colors[i]
        mat_name = f"freq_mat_{i}"
        freq_mat = freq_matrices[method]["Frequency"]
        freq_mat = freq_mat/freq_mat.values.max()
        locals()[mat_name] = freq_mat.values.flatten()

        sns.ecdfplot(locals()[mat_name], label=method, ax=ax, color=color,␣
↪linewidth=2)

    ax.set_xlabel(f"Frequency", fontsize=15)
    ax.set_ylabel("Probability of occurrence", fontsize=15)
    ax.tick_params(axis='both', which='major', labelsize=12)
    patches = [
                mpatches.Patch(color=mcolors.to_rgba(colors[i]), label=name)
                for i, name in enumerate(method_keys.keys())
            ]
    fig.legend(handles=patches, bbox_to_anchor=(0.4, 0.02, 0.5, 0.5))

    # Kruskal-Wallis H-test
    h_statistic, p_value = kruskal(locals()[f"freq_mat_0"],␣
↪locals()[f"freq_mat_1"], locals()[f"freq_mat_2"], locals()[f"freq_mat_3"],␣
↪locals()[f"freq_mat_4"])

    k = 7  # number of groups
    df = k - 1

    ax.text(0.97, 0.08, f"H-statistic: {h_statistic:.1f}\np-value: {p_value:.
↪1e}", fontsize=12, bbox=dict(facecolor='gray', alpha=0.1),
        transform=ax.transAxes, horizontalalignment='right')

    fig.suptitle("ECDF of frequency of significant LR pairs", fontsize=18,␣
↪weight='bold')
    plt.show()
    fig.savefig(f'/Users/sabrina/Library/CloudStorage/
↪OneDrive-UniversityofCopenhagen/Thesis/Figures/Final/F2_Full.svg',␣
↪format='svg', dpi=300, bbox_inches='tight')
```

```
    # Dunn's test
    all_data = [locals()[f"freq_mat_0"], locals()[f"freq_mat_1"],␣
↪locals()[f"freq_mat_2"], locals()[f"freq_mat_3"], locals()[f"freq_mat_4"],␣
↪locals()[f"freq_mat_5"], locals()[f"freq_mat_6"]]
    dunn_results = sp.posthoc_dunn(all_data, p_adjust='bonferroni')
    group_labels = []
    for method in method_keys.keys():
        group_labels.append(method)
    dunn_results.columns = group_labels
    dunn_results.index = group_labels
    significant_pairs = pd.DataFrame()
    non_significant_pairs = pd.DataFrame()

    dunn_results_s = dunn_results[dunn_results < 0.05]
    dunn_results_s = dunn_results_s.stack().reset_index()
    dunn_results_s.columns = ['Group1', 'Group2', 'p-value']
    significant_pairs = pd.concat([significant_pairs, dunn_results_s],␣
↪ignore_index=True)

    dunn_results_n = dunn_results[dunn_results >= 0.05]
    dunn_results_n = dunn_results_n.stack().reset_index()
    dunn_results_n.columns = ['Group1', 'Group2', 'p-value']
    non_significant_pairs = pd.concat([non_significant_pairs, dunn_results_n],␣
↪ignore_index=True)

    significant_pairs = significant_pairs[significant_pairs['Group1'] !=␣
↪significant_pairs['Group2']]
    non_significant_pairs =␣
↪non_significant_pairs[non_significant_pairs['Group1'] !=␣
↪non_significant_pairs['Group2']]
    unique_significant_pairs_df = significant_pairs.groupby(['Group1',␣
↪'Group2']).first().reset_index()
    unique_non_significant_pairs_df = non_significant_pairs.groupby(['Group1',␣
↪'Group2']).first().reset_index()

    print("\nunique significant")
    print(unique_significant_pairs_df)

    print("\nunique non-significant")
    print(unique_non_significant_pairs_df)

    tri = np.triu(dunn_results)
    np.fill_diagonal(tri, False)
    fig, ax = plt.subplots(1, 1, figsize=(10, 8))
    sm = sns.heatmap(
        dunn_results,
```

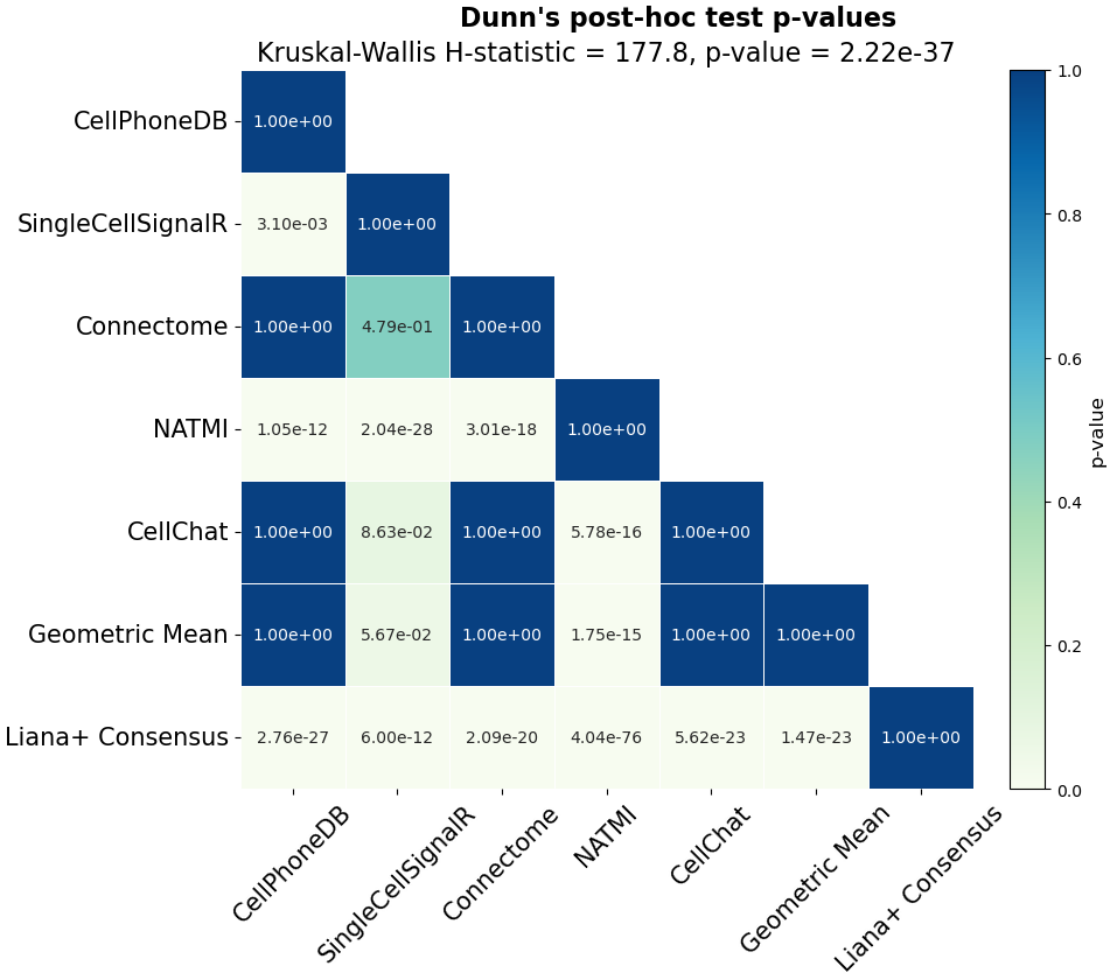
```
        annot=True,
        cmap='GnBu',
        fmt=".2e",
        linewidths=.5,
        vmin=0,
        vmax=1,
        ax=ax,
        mask=tri,
        cbar=False,
    )
    cbar = plt.colorbar(sm.collections[0], ax=ax, orientation='vertical', pad=0.
↪04)
    cbar.set_label('p-value', rotation=90, fontsize=12)

    plt.suptitle("Dunn's post-hoc test p-values\n", fontsize=16, weight='bold',␣
↪y = 0.95)
    ax.set_title(f"Kruskal-Wallis H-statistic = {h_statistic:.1f}, p-value =␣
↪{p_value:.2e}", fontsize=16)
    ax.tick_params(axis='both', labelsize=15)
    ax.tick_params(axis='x', rotation=45)
    plt.show()
    fig.savefig('/Users/sabrina/Library/CloudStorage/
↪OneDrive-UniversityofCopenhagen/Thesis/Figures/Final/dunn_similarity.svg',␣
↪format='svg', dpi=300, bbox_inches='tight')
distribution_comparison(freq_matrices["magnitude"], method_keys)
```



unique significant

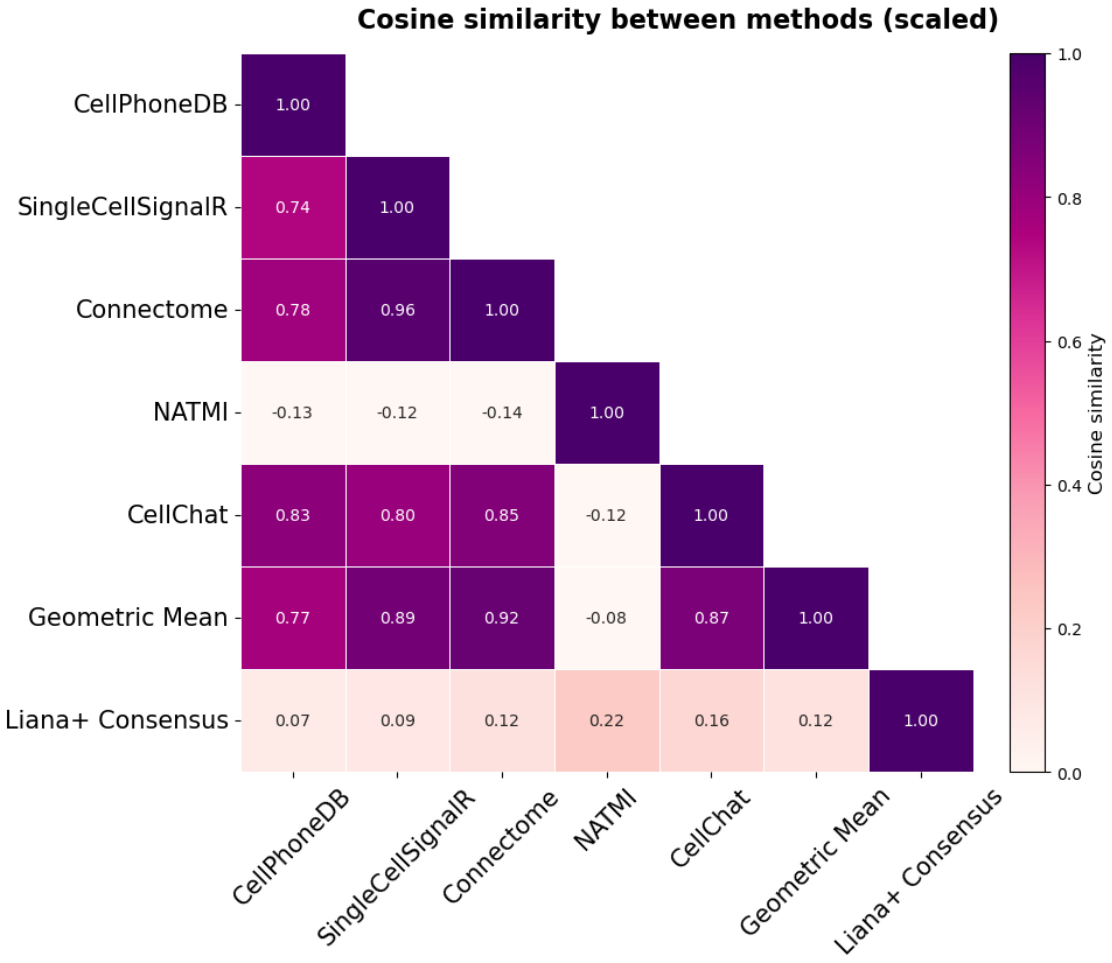
	Group1	Group2	p-value
0	CellChat	Liana+ Consensus	5.619589e-23
1	CellChat	NATMI	5.778545e-16
2	CellPhoneDB	Liana+ Consensus	2.755954e-27
3	CellPhoneDB	NATMI	1.053200e-12
4	CellPhoneDB	SingleCellSignalR	3.101260e-03
5	Connectome	Liana+ Consensus	2.085360e-20
6	Connectome	NATMI	3.006119e-18
7	Geometric Mean	Liana+ Consensus	1.465477e-23
8	Geometric Mean	NATMI	1.747757e-15
9	Liana+ Consensus	CellChat	5.619589e-23
10	Liana+ Consensus	CellPhoneDB	2.755954e-27
11	Liana+ Consensus	Connectome	2.085360e-20
12	Liana+ Consensus	Geometric Mean	1.465477e-23
13	Liana+ Consensus	NATMI	4.039289e-76
14	Liana+ Consensus	SingleCellSignalR	5.997265e-12
15	NATMI	CellChat	5.778545e-16
16	NATMI	CellPhoneDB	1.053200e-12
17	NATMI	Connectome	3.006119e-18
18	NATMI	Geometric Mean	1.747757e-15
19	NATMI	Liana+ Consensus	4.039289e-76

```
20              NATMI  SingleCellSignalR  2.037971e-28
21  SingleCellSignalR        CellPhoneDB  3.101260e-03
22  SingleCellSignalR   Liana+ Consensus  5.997265e-12
23  SingleCellSignalR              NATMI  2.037971e-28

unique non-significant
               Group1             Group2   p-value
0            CellChat        CellPhoneDB  1.000000
1            CellChat         Connectome  1.000000
2            CellChat     Geometric Mean  1.000000
3            CellChat  SingleCellSignalR  0.086250
4         CellPhoneDB           CellChat  1.000000
5         CellPhoneDB         Connectome  1.000000
6         CellPhoneDB     Geometric Mean  1.000000
7          Connectome           CellChat  1.000000
8          Connectome        CellPhoneDB  1.000000
9          Connectome     Geometric Mean  1.000000
10         Connectome  SingleCellSignalR  0.479013
11     Geometric Mean           CellChat  1.000000
12     Geometric Mean        CellPhoneDB  1.000000
13     Geometric Mean         Connectome  1.000000
14     Geometric Mean  SingleCellSignalR  0.056714
15  SingleCellSignalR           CellChat  0.086250
16  SingleCellSignalR         Connectome  0.479013
17  SingleCellSignalR     Geometric Mean  0.056714
```



[ ]:
```python
def calculate_and_plot_cosine_similarity(freq_matrices, method_keys):
    from sklearn.metrics.pairwise import cosine_similarity
    from sklearn.preprocessing import StandardScaler

    vectors = []
    scaler = StandardScaler()
    for method in method_keys.keys():
        matrix = freq_matrices[method]['Frequency']
        scaled_data = scaler.fit_transform(matrix)
        vectors.append(np.array(scaled_data).flatten())

    final_matrix = np.vstack(vectors)

    similarities = cosine_similarity(final_matrix)

    tri = np.triu(similarities)
```

```
    np.fill_diagonal(tri, False)
    fig, ax = plt.subplots(1, 1, figsize=(10, 8))
    h = sns.heatmap(
        similarities,
        xticklabels=list(method_keys.keys()),
        yticklabels=list(method_keys.keys()),
        annot=True,
        cmap='RdPu',
        fmt=".2f",
        linewidths=.5,
        vmin=0,
        vmax=1,
        ax=ax,
        mask=tri,
        cbar=False,
    )
    cbar = plt.colorbar(h.collections[0], ax=ax, orientation='vertical', pad=0.
↪04)
    cbar.set_label('Cosine similarity', rotation=90, fontsize=12)

    plt.suptitle('Cosine similarity between methods (scaled)', fontsize=16,␣
↪weight='bold', y = 0.93)
    ax.tick_params(axis='both', labelsize=15)
    ax.tick_params(axis='x', rotation=45)
    plt.show()
    fig.savefig('/Users/sabrina/Library/CloudStorage/
↪OneDrive-UniversityofCopenhagen/Thesis/Figures/Final/cosine_similarity.svg',␣
↪format='svg', dpi=300, bbox_inches='tight')

calculate_and_plot_cosine_similarity(freq_matrices["magnitude"], method_keys)
```



4 Frequency of interaction fails to predict evidence-based connectivity

[ ]:
```
def plot_survey_pairs_ecdf(freq_matrices, data_name, method_keys,␣
↪survey_pairs=pd.read_csv('../survey_pairs.csv', index_col=0).T):
    from matplotlib import colors as mcolors
    from matplotlib import patches as mpatches
    from scipy.stats import kruskal
    import scikit_posthocs as sp

    colors = ["#D81B60", "#1E88E5","#FFC107", "#004D40", "#E68CB4"]
    labels = ['Anatomical/physiological evidence', 'Anatomical/physiological␣
↪support', 'Predicted', 'Low/absent support', 'Unknown interconnectivity']

    fig, axs = plt.subplots(2, 4, figsize=(20, 6))
```

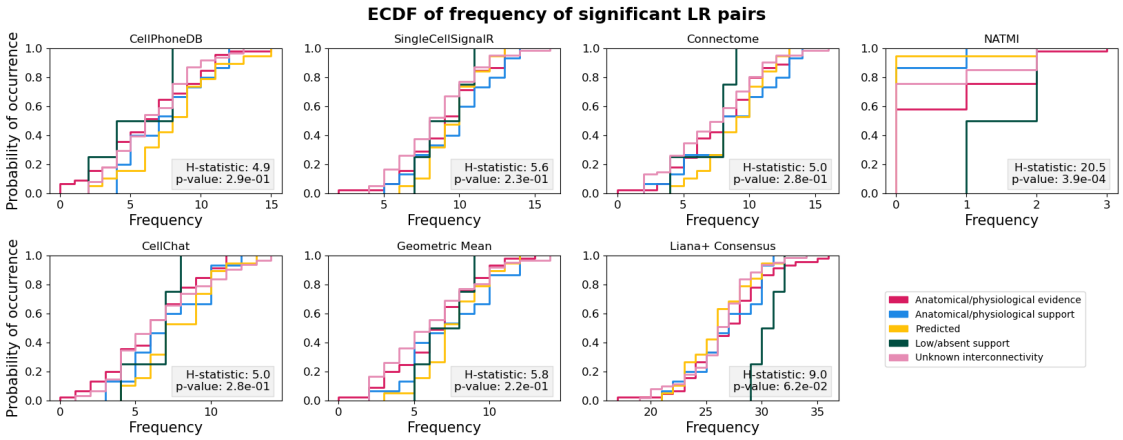
```
    axs = axs.flatten()
    fig.subplots_adjust(hspace=0.2, wspace=0.2)
    significant_pairs = pd.DataFrame(columns=['Group1', 'Group2', 'p-value',␣
↪'Method'])
    for i, method in enumerate(method_keys.keys()):
        ax = axs[i]
        freq_mat = freq_matrices[method]["Frequency"]
        if all(survey_pairs.index == freq_mat.index) and all(survey_pairs.
↪columns == freq_mat.columns):
            ev_mat = freq_mat[survey_pairs == "evidence"].values.flatten()
            ev_mat = ev_mat[~np.isnan(ev_mat)]
            unknown_mat = freq_mat[survey_pairs == "unknown"].values.flatten()
            unknown_mat = unknown_mat[~np.isnan(unknown_mat)]
            sup_mat = freq_mat[survey_pairs == "support"].values.flatten()
            sup_mat = sup_mat[~np.isnan(sup_mat)]
            pred_mat = freq_mat[survey_pairs == "predicted"].values.flatten()
            pred_mat = pred_mat[~np.isnan(pred_mat)]
            low_mat = freq_mat[survey_pairs == "low/absent"].values.flatten()
            low_mat = low_mat[~np.isnan(low_mat)]
        else:
            print("Error: survey_pairs does not match freq_mat order.")
        #total = sum(survey_pairs == "low/absent") + sum(survey_pairs ==␣
↪"predicted") + sum(survey_pairs == "support") + sum(survey_pairs ==␣
↪"evidence") + sum(survey_pairs == "unknown")
        #print(total)

        sns.ecdfplot(ev_mat, color=colors[0], ax=ax, linewidth=2)
        sns.ecdfplot(sup_mat, color=colors[1], ax=ax, linewidth=2)
        sns.ecdfplot(pred_mat, color=colors[2], ax=ax, linewidth=2)
        sns.ecdfplot(low_mat, color=colors[3], ax=ax, linewidth=2)
        sns.ecdfplot(unknown_mat, color=colors[4], ax=ax, linewidth=2)

        ax.set_title(method)
        ax.set_xlabel(f"Frequency", fontsize=15)
        ax.set_ylabel("Probability of occurrence", fontsize=15) if i == 0 or i␣
↪== 4 else ax.set_ylabel("")
        ax.tick_params(axis='both', labelsize=12)
        if i == 0:
            patches = [
                mpatches.Patch(color=mcolors.to_rgba(colors[i]), label=name)
                for i, name in enumerate(labels)
            ]
            fig.legend(handles=patches, loc='lower right', bbox_to_anchor=(0.
↪38, 0.1, 0.5, 0.5))

        # Kruskal-Wallis H-test
```

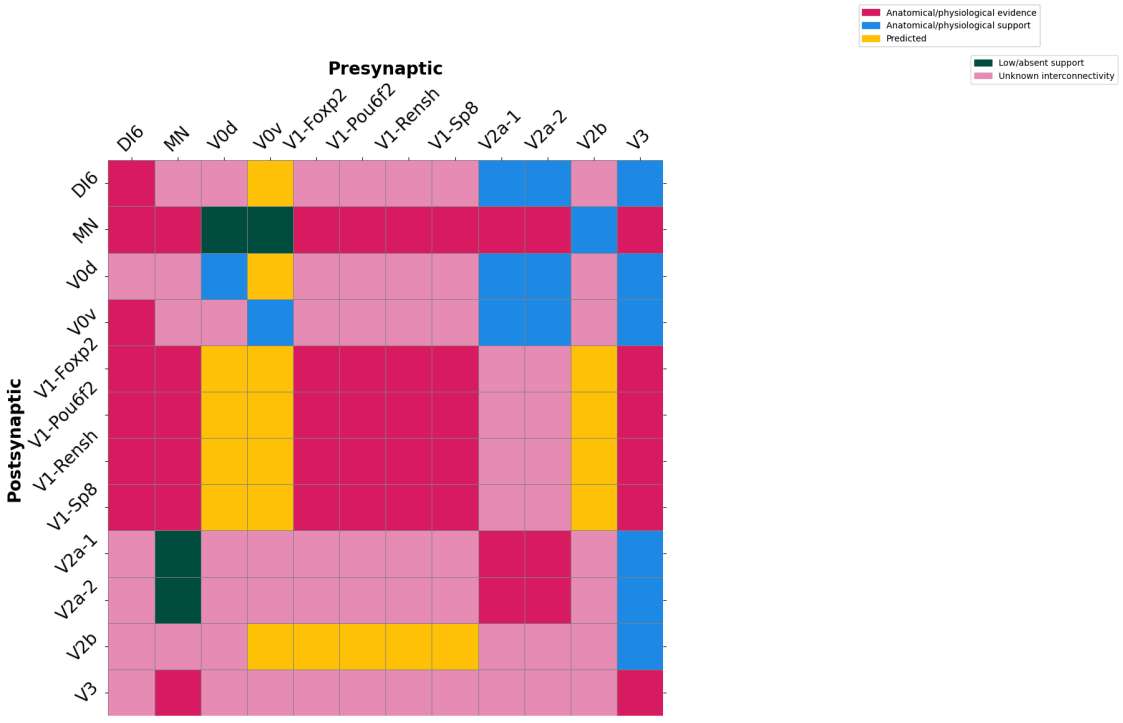
```
        h_statistic, p_value = kruskal(ev_mat, sup_mat, pred_mat, low_mat,␣
↪unknown_mat)

        k = 5  # number of groups
        df = k - 1

        if i >= 4:
            pos = ax.get_position()
            axs[i].set_position([
                pos.x0,
                pos.y0 - 0.08,
                pos.width,
                pos.height,
            ])
        ax.text(0.95, 0.07, f"H-statistic: {h_statistic:.1f}\np-value: {p_value:
↪.1e}", fontsize=12, bbox=dict(facecolor='gray', alpha=0.1),
            transform=ax.transAxes, horizontalalignment='right')

        all_data = [ev_mat, sup_mat, pred_mat, low_mat, unknown_mat]
        dunn_results = sp.posthoc_dunn(all_data, p_adjust='bonferroni')
        dunn_results.columns = labels
        dunn_results.index = labels

        if any(dunn_results.values.flatten() < 0.05):
            dunn_results = dunn_results[dunn_results < 0.05]
            dunn_results = dunn_results.stack().reset_index()
            dunn_results.columns = ['Group1', 'Group2', 'p-value']
            dunn_results = dunn_results[dunn_results['p-value'] < 0.05]
            dunn_results['Method'] = method
            significant_pairs = pd.concat([significant_pairs, dunn_results],␣
↪ignore_index=True)

    significant_pairs = significant_pairs.sort_values(by=['Group1', 'Group2'])
    significant_pairs.to_csv(f'/Users/sabrina/Library/CloudStorage/
↪OneDrive-UniversityofCopenhagen/Thesis/Figures/
↪ECDF_survey_{data_name}_significant_pairs.csv', index=False)


    axs[-1].set_axis_off()
    fig.suptitle("ECDF of frequency of significant LR pairs", fontsize=18,␣
↪weight='bold')
    plt.show()
    fig.savefig(f'/Users/sabrina/Library/CloudStorage/
↪OneDrive-UniversityofCopenhagen/Thesis/Figures/ECDF{data_name}.svg',␣
↪format='svg', dpi=1200, bbox_inches='tight')
```

```
    """
    ------- Classification heatmap -------
    """

    mask_groups = [
        'evidence',
        'support',
        'predicted',
        'low/absent',
        'unknown'
    ]

    group_to_num = {name: i for i, name in enumerate(mask_groups)}
    cmap = mcolors.ListedColormap(colors)
    numerical_df = survey_pairs.applymap(lambda x: group_to_num.get(x))

    fig, ax = plt.subplots(figsize=(19.20,10.80))

    sns.heatmap(
        numerical_df,
        cmap=cmap,
        linewidths=0.5,
        linecolor='gray',
        square=True,
        cbar=False
    )

    plt.title("Presynaptic", weight='bold', fontsize=20)
    plt.ylabel("Postsynaptic", weight='bold', fontsize=20)

    plt.tick_params(top=True, labeltop=True, bottom=False, labelbottom=False)
    plt.tick_params(axis='both', rotation=45, right=True, labelsize=20)
    patches1 = [
                mpatches.Patch(color=mcolors.to_rgba(colors[i]), label=name)
                for i, name in zip(range(3), labels[0:3])
            ]
    patches2 = [
                mpatches.Patch(color=mcolors.to_rgba(colors[i]), label=name)
                for i, name in zip(range(3, 5), labels[3:5])
            ]

    l1 = fig.legend(handles=patches1, bbox_to_anchor=(1.05, 1), loc='lower␣
↪right')
    fig.legend(handles=patches2, bbox_to_anchor=(1.05, 1), loc='upper center')
    fig.add_artist(l1)
    plt.tight_layout()
    plt.show()
```

```
    fig.savefig('/Users/sabrina/Library/CloudStorage/
↪OneDrive-UniversityofCopenhagen/Thesis/Figures/classification.svg',␣
↪format='svg', dpi=300, bbox_inches='tight')
```

[193]:
```
plot_survey_pairs_ecdf(freq_matrices['magnitude'], "full", method_keys,␣
↪survey_pairs=survey_pairs)
```



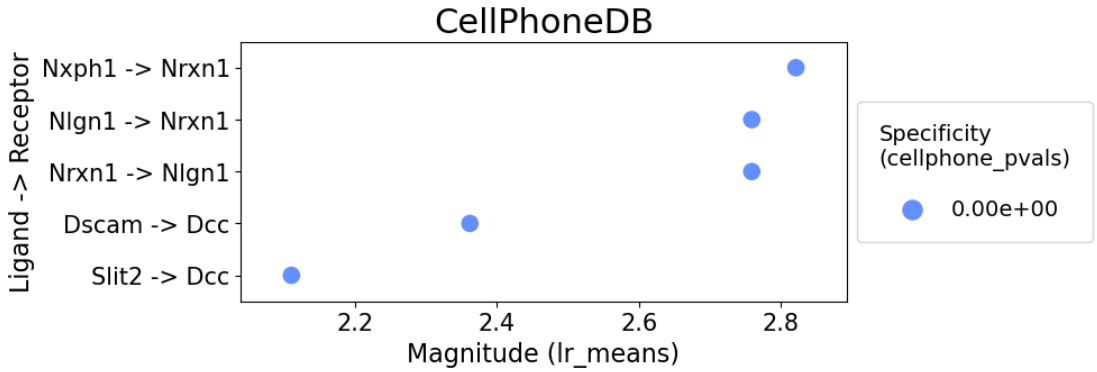


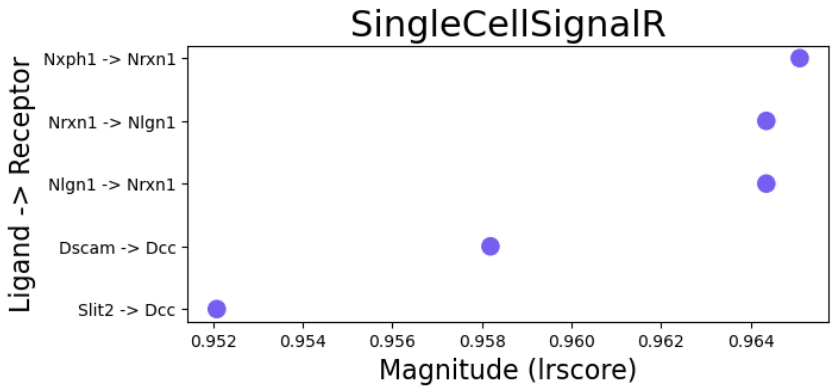
5 Gene analysis reveals a structural basis for connectivity

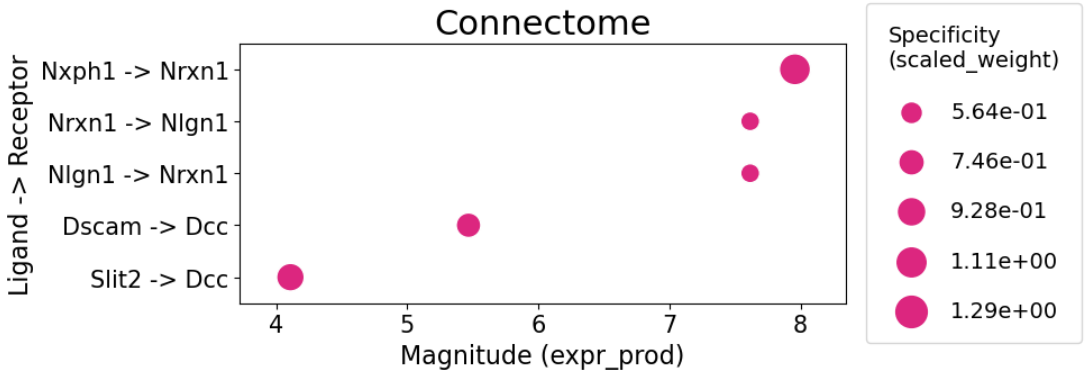
[ ]:
```
def plot_top_n_lr_pairs(df, method_key, color, top_n, data_name, figsize = (10,␣
↪3), size_scale = (150, 400)):
    colors = ["#648FFF", "#785EF0","#DC267F", "#FE6100", "#FFB000", "#CFA8DB",␣
↪"#AE19C2"]
    color = colors[color]

    fig = plt.figure(figsize=figsize)
    gs = gridspec.GridSpec(1, 2, width_ratios=[1, 0.25], wspace=0.3)
    ax = fig.add_subplot(gs[0, 0])
    side_ax = fig.add_subplot(gs[0, 1])
    side_ax.set_axis_off()

    method, key, mag_col, spec_col = method_key

    df['interaction'] = df['ligand_complex'] + ' -> ' + df['receptor_complex']
    top_interactions = pd.DataFrame()

    if method in ['CellPhoneDB', 'CellChat', 'Geometric Mean']:
        df = df.sort_values(by=[mag_col, spec_col], ascending=[False, True])
        df = df[~df['interaction'].duplicated(keep='first')]
        top_interactions = df.head(top_n)
    elif method == 'SingleCellSignalR':
        df = df.sort_values(by=mag_col, ascending=False)
        df = df[~df['interaction'].duplicated(keep='first')]
        top_interactions = df.head(top_n)
    elif method == 'Connectome':
        df = df.sort_values(by=[mag_col, spec_col], ascending=[False, False])
        df = df[~df['interaction'].duplicated(keep='first')]
        top_interactions = df.head(top_n)
    elif method == 'NATMI':
        df = df.sort_values(by=[mag_col, spec_col], ascending=[False, False])
        df = df[~df['interaction'].duplicated(keep='first')]
        top_interactions = df.head(top_n)
    elif method == 'Liana+ Consensus':
        df = df.sort_values(by=[mag_col, spec_col], ascending=[True, True])
        df = df[~df['interaction'].duplicated(keep='first')]
        top_interactions = df.head(top_n)
    else:
        print(f"Warning: No specific sorting logic for method '{method}'.␣
↪Skipping.")

    top_interaction_names = top_interactions['interaction'].tolist()
    plot_data = df[df['interaction'].isin(top_interaction_names)].copy()

    if method != 'SingleCellSignalR':
```

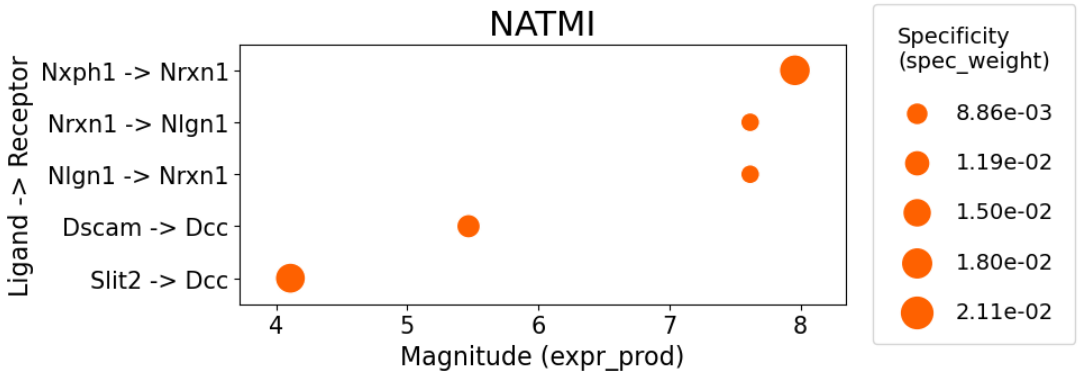
```
        spec_scores = plot_data[spec_col]
        min_spec, max_spec = spec_scores.min(), spec_scores.max()
        if min_spec == max_spec:
            plot_data['dot_size'] = size_scale[0]
        else:
            plot_data['dot_size'] = (
                ((spec_scores - min_spec) / (max_spec - min_spec))
                * (size_scale[1] - size_scale[0])
                + size_scale[0]
            )

        scat_plot = sns.scatterplot(
            data=plot_data,
            y='interaction',
            x=mag_col,
            ax=ax,
            size='dot_size',
            color=color,
            sizes=size_scale,
            legend=False,
        )

        legend_sizes = np.linspace(size_scale[0], size_scale[1], 5)
        legend_labels = np.linspace(spec_scores.min(), spec_scores.max(), 5)
        handles = []
        for i, s in enumerate(legend_sizes):
            handles.append(ax.scatter([], [], s=s, color=color,␣
↪label=f"{legend_labels[i]:.2e}", alpha=1))

        legend = side_ax.legend(
            handles=handles[0:1] if method in ['CellPhoneDB', "CellChat",␣
↪"Geometric Mean"] else handles,
            title=f'Specificity\n({spec_col})',
            loc='center',
            bbox_to_anchor=(0.1, 0.5), labelspacing=1.3, borderpad=1,␣
↪fontsize=14, title_fontsize=14
        )

        dot_x_values = plot_data[mag_col].values
        min_x, max_x = dot_x_values.min(), dot_x_values.max()
        x_range = max_x - min_x if max_x > min_x else 1
        ax.set_xlim(min_x - 0.1 * x_range, max_x + 0.1 * x_range)
        ax.set_ylim(4.5, -0.5)
        ax.tick_params(axis='both', which='major', labelsize=15)


    else:
```

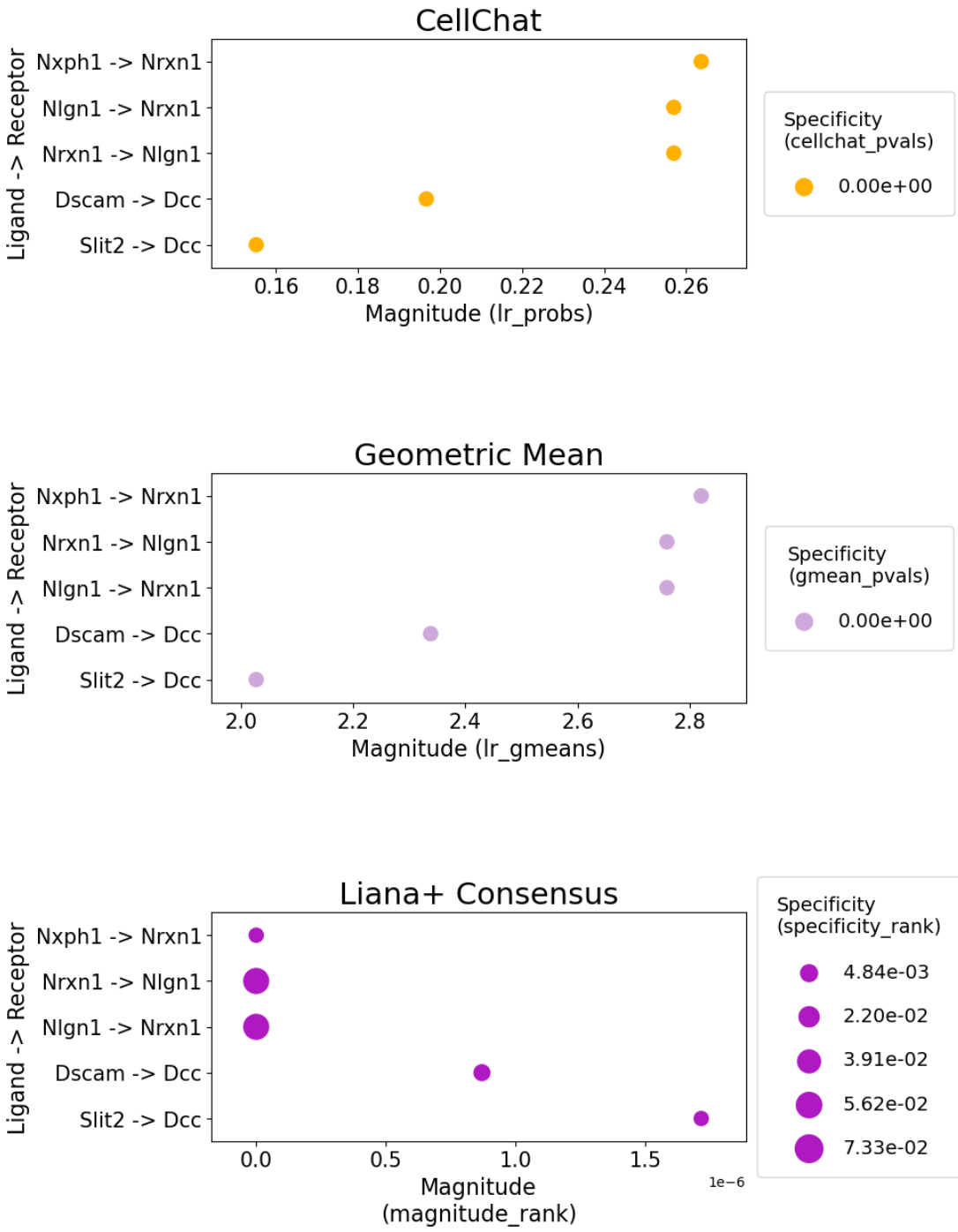
```
        scat_plot = sns.scatterplot(
            data=plot_data,
            y='interaction',
            x=mag_col,
            ax=ax,
            color=color,
            size=1,
            sizes=size_scale,
            legend=False
        )

    ax.set_title(f"{method}", fontsize=22)
    ax.set_xlabel(f'Magnitude\n({mag_col})' if method == "Liana+ Consensus"␣
↪else f'Magnitude ({mag_col})', fontsize=16)
    ax.set_ylabel('Ligand -> Receptor', fontsize=16)

    plt.tight_layout()
    plt.show()
    fig.savefig(f'/Users/sabrina/Library/CloudStorage/
↪OneDrive-UniversityofCopenhagen/Thesis/Figures/genes_{method}_{data_name}.
↪svg', format='svg', dpi=300, bbox_inches='tight')

for i, (method, (key, mag_col, spec_col)) in zip(range(7), method_keys.items()):
    plot_top_n_lr_pairs(adata.uns[key], method_key=(method, key, mag_col,␣
↪spec_col), top_n=5, color=i, data_name='Full data')
    #plot_top_n_lr_pairs(adata_filt.uns[key], method_key=(method, key, mag_col,␣
↪spec_col), top_n=5, color=i, data_name='Filtered data')
```




SingleCellSignalR
Ligand -> Receptor
Nxph1 -> Nrxn1
Nrxn1 -> Nlgn1
Nlgn1 -> Nrxn1
Dscam -> Dcc
Slit2 -> Dcc
0.952
0.954
0.956
0.958
0.960
0.962
0.964
Magnitude (lrscore)


Connectome
Ligand -> Receptor
Nxph1 -> Nrxn1
Nrxn1 -> Nlgn1
Nlgn1 -> Nrxn1
Dscam -> Dcc
Slit2 -> Dcc
4
5
6
7
8
Magnitude (expr_prod)
Specificity
(scaled_weight)
5.64e-01
7.46e-01
9.28e-01
1.11e+00
1.29e+00


NATMI
Ligand -> Receptor
Nxph1 -> Nrxn1
Nrxn1 -> Nlgn1
Nlgn1 -> Nrxn1
Dscam -> Dcc
Slit2 -> Dcc
4
5
6
7
8
Magnitude (expr_prod)
Specificity
(spec_weight)
8.86e-03
1.19e-02
1.50e-02
1.80e-02
2.11e-02

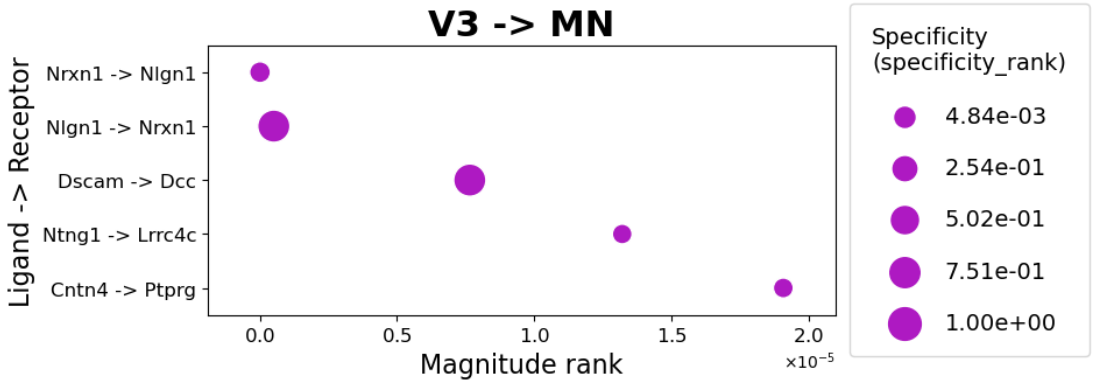


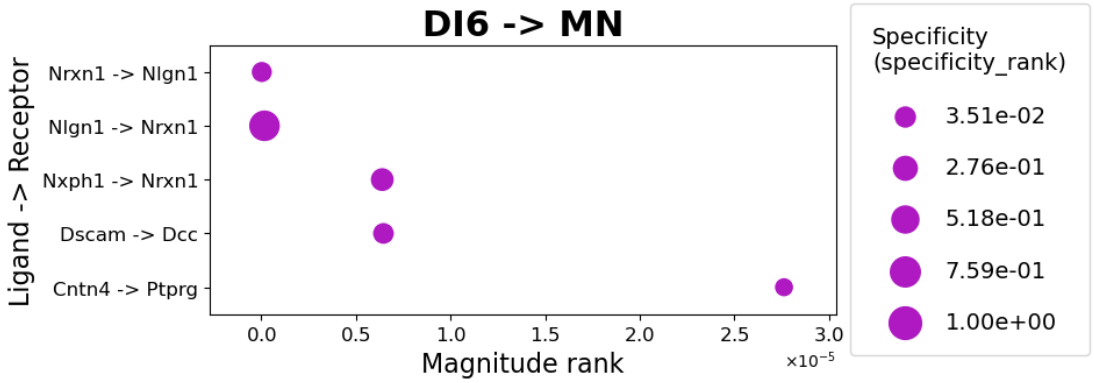
[ ]:
```
def plot_top_n_lr_CT_pairs(df, method_key, color, top_n, source, target,␣
↪data_name, figsize = (10, 3), size_scale = (150, 400)):
    from matplotlib.ticker import ScalarFormatter
    colors = ["#648FFF", "#785EF0","#DC267F", "#FE6100", "#FFB000", "#CFA8DB",␣
↪"#AE19C2"]
```

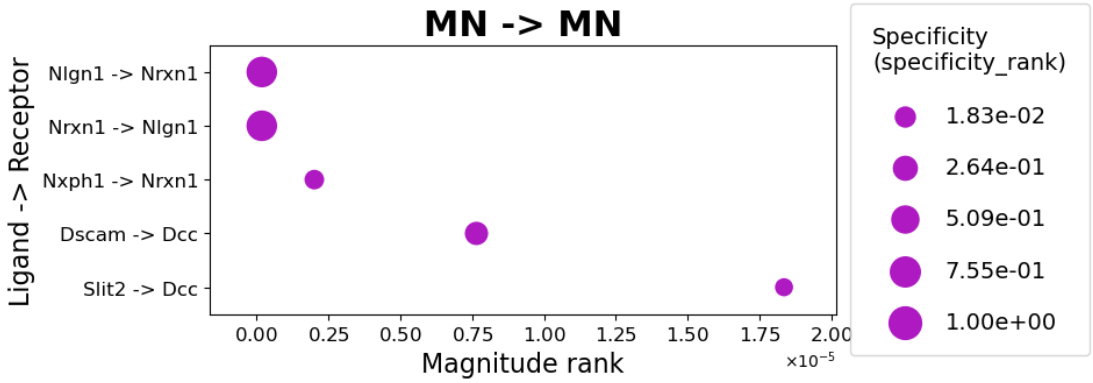
```
    color = colors[color]

    fig = plt.figure(figsize=figsize)
    gs = gridspec.GridSpec(1, 2, width_ratios=[1, 0.25], wspace=0.3)
    ax = fig.add_subplot(gs[0, 0])
    side_ax = fig.add_subplot(gs[0, 1])
    side_ax.set_axis_off()

    method, key, mag_col, spec_col = method_key
    # keep only pairs from specific source -> target
    df = df[(df['source'] == source) & (df['target'] == target)]
    df['interaction'] = df['ligand_complex'] + ' -> ' + df['receptor_complex']
    top_interactions = pd.DataFrame()

    df = df.sort_values(by=[mag_col, spec_col], ascending=[True, True])
    df = df[~df['interaction'].duplicated(keep='first')]
    top_interactions = df.head(top_n)

    top_interaction_names = top_interactions['interaction'].tolist()
    plot_data = df[df['interaction'].isin(top_interaction_names)].copy()

    spec_scores = plot_data[spec_col]
    min_spec, max_spec = spec_scores.min(), spec_scores.max()
    if min_spec == max_spec:
        plot_data['dot_size'] = size_scale[0]
    else:
        plot_data['dot_size'] = (
            ((spec_scores - min_spec) / (max_spec - min_spec))
            * (size_scale[1] - size_scale[0])
            + size_scale[0]
        )

    scat_plot = sns.scatterplot(
        data=plot_data,
        y='interaction',
        x=mag_col,
        ax=ax,
        size='dot_size',
        color=color,
        sizes=size_scale,
        legend=False,
    )

    legend_sizes = np.linspace(size_scale[0], size_scale[1], 5)
    legend_labels = np.linspace(spec_scores.min(), spec_scores.max(), 5)
    handles = []
    for i, s in enumerate(legend_sizes):
```

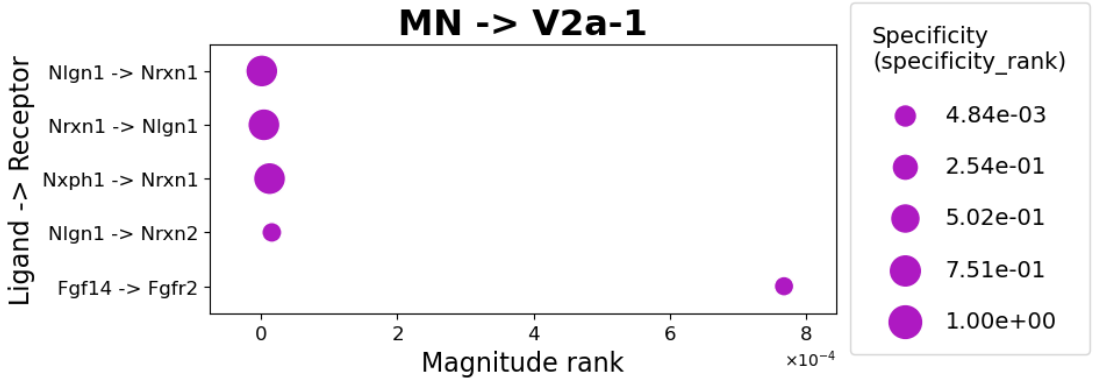
```
        handles.append(ax.scatter([], [], s=s, color=color,␣
↪label=f"{legend_labels[i]:.2e}", alpha=1))

    legend = side_ax.legend(
        handles=handles[0:1] if method in ['CellPhoneDB', "CellChat",␣
↪"Geometric Mean"] else handles,
        title=f'Specificity\n({spec_col})',
        loc='center',
        bbox_to_anchor=(0.1, 0.5), labelspacing=1.3, borderpad=1, fontsize=14,␣
↪title_fontsize=14
    )

    dot_x_values = plot_data[mag_col].values
    min_x, max_x = dot_x_values.min(), dot_x_values.max()
    x_range = max_x - min_x if max_x > min_x else 1
    ax.set_xlim(min_x - 0.1 * x_range, max_x + 0.1 * x_range)
    ax.set_ylim(4.5, -0.5)
    ax.tick_params(axis='both', which='major', labelsize=12)
    ax.xaxis.set_major_formatter(ScalarFormatter(useMathText=True))
    ax.ticklabel_format(style='sci', axis='x', scilimits=(0,0))


    ax.set_title(f"{source} -> {target}", fontsize=22, weight = 'bold')
    ax.set_xlabel(f'Magnitude rank', fontsize=16)
    ax.set_ylabel('Ligand -> Receptor', fontsize=16)

    plt.tight_layout()
    plt.show()
    fig.savefig(f'/Users/sabrina/Library/CloudStorage/
↪OneDrive-UniversityofCopenhagen/Thesis/Figures/
↪CTgenes_{source}-{target}_{data_name}.svg', format='svg', dpi=300,␣
↪bbox_inches='tight')


for i, (method, (key, mag_col, spec_col)) in zip(range(7), method_keys.items()):
    if method == 'Liana+ Consensus':
        plot_top_n_lr_CT_pairs(adata.uns[key], method_key=(method, key,␣
↪mag_col, spec_col), top_n=5, color=i, source="V3", target="MN",␣
↪data_name="Full")
        plot_top_n_lr_CT_pairs(adata.uns[key], method_key=(method, key,␣
↪mag_col, spec_col), top_n=5, color=i, source="DI6", target="MN",␣
↪data_name="Full")
        plot_top_n_lr_CT_pairs(adata.uns[key], method_key=(method, key,␣
↪mag_col, spec_col), top_n=5, color=i, source="MN", target="MN",␣
↪data_name="Full")
```

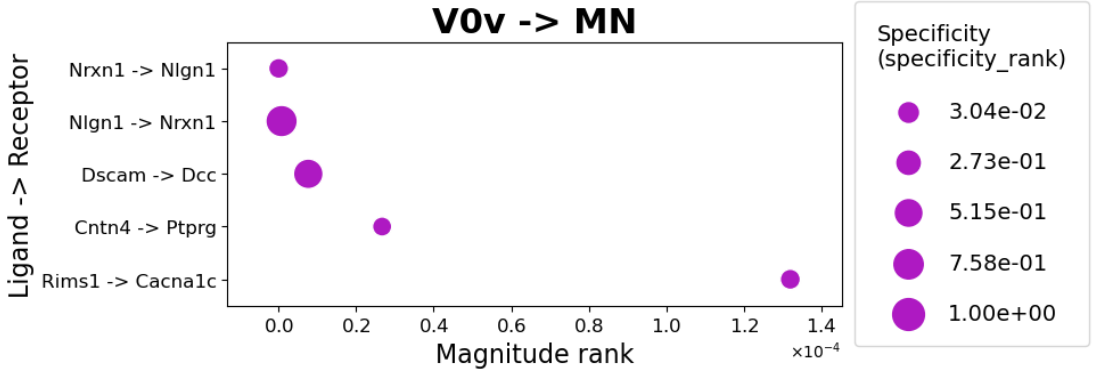
```
      plot_top_n_lr_CT_pairs(adata.uns[key], method_key=(method, key,␣
↪mag_col, spec_col), top_n=5, color=i, source="MN", target="V2a-1",␣
↪data_name="Full")
      plot_top_n_lr_CT_pairs(adata.uns[key], method_key=(method, key,␣
↪mag_col, spec_col), top_n=5, color=i, source="V0v", target="MN",␣
↪data_name="Full")
      plot_top_n_lr_CT_pairs(adata.uns[key], method_key=(method, key,␣
↪mag_col, spec_col), top_n=5, color=i, source="V0d", target="MN",␣
↪data_name="Full")
```

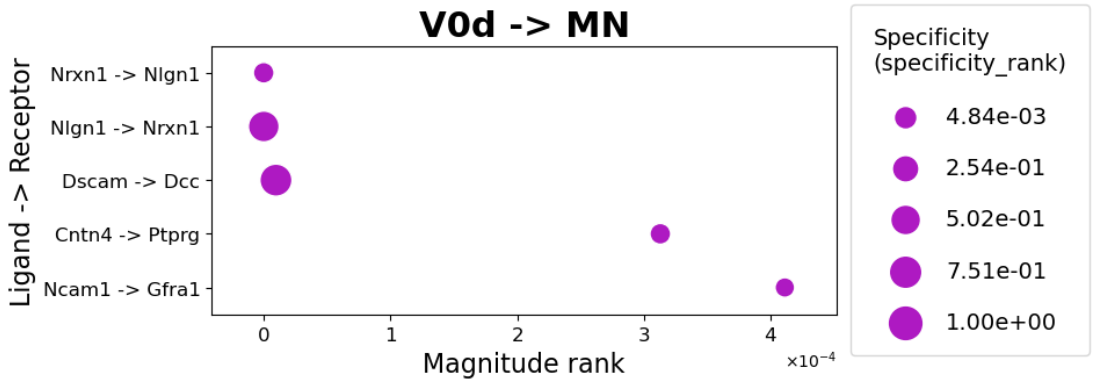





MN -> MN
Ligand -> Receptor
Nlgn1 -> Nrxn1
Nrxn1 -> Nlgn1
Nxph1 -> Nrxn1
Dscam -> Dcc
Slit2 -> Dcc
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Magnitude rank
×10⁻⁵
Specificity
(specificity_rank)
1.83e-02
2.64e-01
5.09e-01
7.55e-01
1.00e+00


MN -> V2a-1
Ligand -> Receptor
Nlgn1 -> Nrxn1
Nrxn1 -> Nlgn1
Nxph1 -> Nrxn1
Nlgn1 -> Nrxn2
Fgf14 -> Fgfr2
0
2
4
6
8
Magnitude rank
×10⁻⁴
Specificity
(specificity_rank)
4.84e-03
2.54e-01
5.02e-01
7.51e-01
1.00e+00


V0v -> MN
Ligand -> Receptor
Nrxn1 -> Nlgn1
Nlgn1 -> Nrxn1
Dscam -> Dcc
Cntn4 -> Ptprg
Rims1 -> Cacna1c
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
Magnitude rank
×10⁻⁴
Specificity
(specificity_rank)
3.04e-02
2.73e-01
5.15e-01
7.58e-01
1.00e+00



[184]:
```
def topLR_byCTpair(adata, freq_mat, percent):
    liana_df = adata.uns['liana_res']
    unique_top_lr = {}

    for source in freq_mat.columns:
        for target in freq_mat.index:
            pair_df = liana_df[(liana_df['source'] == source) &␣
↪(liana_df['target'] == target)]
            pair_df['lr_pair'] = pair_df['ligand_complex'] + ' -> ' +␣
↪pair_df['receptor_complex']
            all_lr = liana_df['ligand_complex'] + ' -> ' +␣
↪liana_df['receptor_complex']
            total_pairs = len(freq_mat.columns) * len(freq_mat.index)
            if percent < 1:
                threshold = max(1, int(total_pairs * percent))
            elif percent == 1:
                threshold = 1
            unique_lr = pair_df[pair_df['lr_pair'].map(lambda x: (all_lr == x).
↪sum() <= threshold)]
            if not unique_lr.empty:
                top_lrs = unique_lr.sort_values('magnitude_rank').head(5)
                unique_top_lr[(source, target)] = top_lrs['lr_pair'].values.
↪tolist()
            else:
                unique_top_lr[(source, target)] = None

    ct_matrix = pd.DataFrame(index=freq_mat.columns, columns=freq_mat.index)

    for (source, target), lr in unique_top_lr.items():
        ct_matrix.loc[target, source] = lr # rows are targets, columns are␣
↪sources
```

```
    if percent < 1:
        ct_matrix.to_csv(f"unique_top_lr_pairs_{percent}percent.csv")
    elif percent == 1:
        ct_matrix.to_csv(f"unique_top_lr_pairs_only1.csv")

    unique_top_lr = {k: v for k, v in unique_top_lr.items() if v is not None}
    print(unique_top_lr)
    unique_genes = set()

    for gene_pairs_list in unique_top_lr.values():
        for pair_string in gene_pairs_list:
            parts = pair_string.split(' -> ')
            genes_left = parts[0].split('_')
            for gene in genes_left:
                unique_genes.add(gene.strip())

            genes_right = parts[1].split('_')
            for gene in genes_right:
                unique_genes.add(gene.strip())

    with open(f"../CTpairs_unique_genes_{percent}.txt", "w") as f:
        f.write("\n".join(unique_genes))

topLR_byCTpair(adata, freq_matrices['magnitude']['Liana+␣
 ↪Consensus']["Frequency"], percent=0.05)
topLR_byCTpair(adata, freq_matrices['magnitude']['Liana+␣
 ↪Consensus']["Frequency"], percent=1)
```

```
{('DI6', 'DI6'): ['F8 -> Lrp2', 'Tctn1 -> Tmem67'], ('DI6', 'V1-Rensh'): ['Bsg
-> Slc16a1', 'Tctn1 -> Tmem67'], ('DI6', 'V2b'): ['F8 -> Lrp2'], ('MN', 'DI6'):
['Rgma -> Bmpr1b', 'Tctn1 -> Tmem67'], ('MN', 'MN'): ['Tgfb2 -> Acvr1_Tgfbr2',
'Sema3d -> Nrp1_Plxna3', 'Sema3d -> Nrp2_Plxna3', 'Rgma -> Bmpr1b', 'Lama3 ->
Itga7_Itgb1'], ('MN', 'V0v'): ['Sema3d -> Nrp1_Plxna3', 'Sema3d -> Nrp2_Plxna3',
'Rgma -> Bmpr1b'], ('MN', 'V1-Pou6f2'): ['Sema3d -> Nrp1_Plxna3', 'Sema3d ->
Nrp2_Plxna3', 'Rgma -> Bmpr1b'], ('MN', 'V1-Rensh'): ['Tctn1 -> Tmem67'], ('MN',
'V2a-1'): ['Kitl -> Kit'], ('MN', 'V2a-2'): ['Rgma -> Bmpr1b'], ('MN', 'V2b'):
['Sema3d -> Nrp1_Plxna3', 'Sema3d -> Nrp2_Plxna3', 'Rgma -> Bmpr1b'], ('MN',
'V3'): ['Sema3d -> Nrp1_Plxna3', 'Sema3d -> Nrp2_Plxna3', 'Rgma -> Bmpr1b'],
('V0d', 'DI6'): ['Lama2 -> Itga9_Itgb1'], ('V0d', 'MN'): ['Lama2 ->
Itga7_Itgb1', 'Lama2 -> Itga6_Itgb1', 'Lama2 -> Itga9_Itgb1'], ('V0d',
'V1-Foxp2'): ['Lama2 -> Itga9_Itgb1'], ('V0d', 'V1-Pou6f2'): ['Lama2 ->
Itga9_Itgb1'], ('V0d', 'V1-Rensh'): ['Bsg -> Slc16a1', 'Lama2 -> Itga6_Itgb1'],
('V0d', 'V2a-1'): ['Lama2 -> Itga9_Itgb1', 'Lama2 -> Rpsa'], ('V0d', 'V2a-2'):
['Lama2 -> Itga9_Itgb1', 'Lama2 -> Rpsa'], ('V0d', 'V3'): ['Lama2 ->
Itga6_Itgb1', 'Lama2 -> Itga9_Itgb1'], ('V0v', 'DI6'): ['Gpc3 -> Lrp2', 'Penk ->
Oprd1'], ('V0v', 'MN'): ['Lama1 -> Itga7_Itgb1'], ('V0v', 'V1-Foxp2'): ['Penk ->
Oprd1'], ('V0v', 'V1-Pou6f2'): ['Penk -> Oprk1'], ('V0v', 'V1-Rensh'): ['Bsg ->
Slc16a1'], ('V0v', 'V2a-1'): ['Kitl -> Kit', 'Gpc3 -> Cd81', 'Penk -> Ogfr',
```

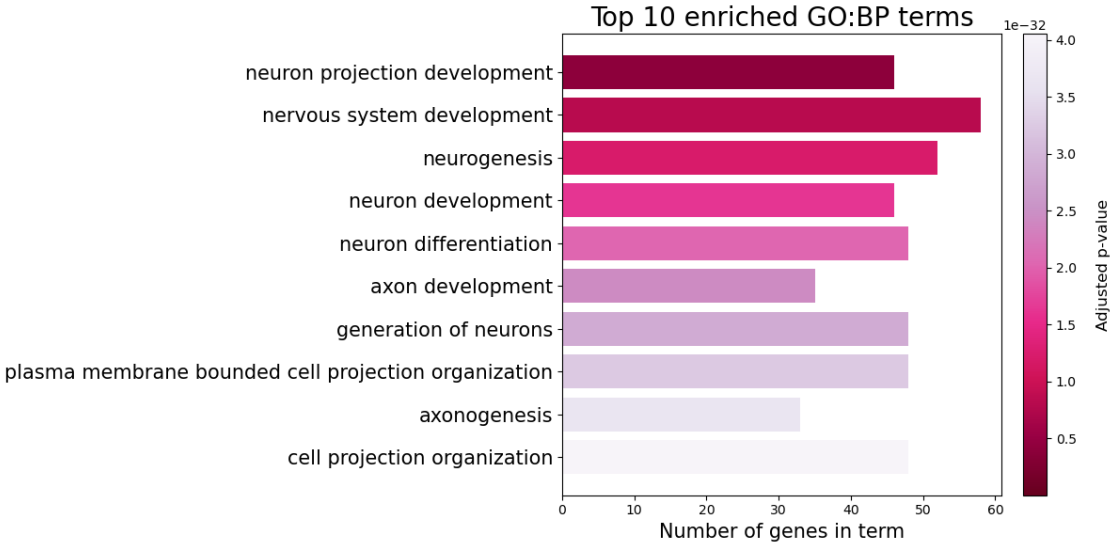
'Penk -> Oprl1'], ('V0v', 'V2a-2'): ['Gpc3 -> Cd81', 'Penk -> Oprd1'], ('V0v', 'V2b'): ['Gpc3 -> Lrp2'], ('V1-Foxp2', 'DI6'): ['F8 -> Lrp2', 'Wnt5b -> Fzd3_Lrp6', 'Tctn1 -> Tmem67'], ('V1-Foxp2', 'V0v'): ['Wnt5b -> Fzd3_Lrp6'], ('V1-Foxp2', 'V1-Pou6f2'): ['Wnt5b -> Fzd3_Lrp6'], ('V1-Foxp2', 'V1-Rensh'): ['Bsg -> Slc16a1', 'Tctn1 -> Tmem67'], ('V1-Foxp2', 'V1-Sp8'): ['Wnt5b -> Fzd3_Lrp6'], ('V1-Foxp2', 'V2a-1'): ['Kitl -> Kit'], ('V1-Foxp2', 'V2a-2'): ['Wnt5b -> Fzd3_Lrp6'], ('V1-Foxp2', 'V2b'): ['F8 -> Lrp2'], ('V1-Foxp2', 'V3'): ['Wnt5b -> Fzd3_Lrp6'], ('V1-Pou6f2', 'MN'): ['Lama1 -> Itga7_Itgb1', 'Ntn1 -> Mcam', 'Tgfb2 -> Acvr1_Tgfbr2'], ('V1-Pou6f2', 'V1-Rensh'): ['Bsg -> Slc16a1'], ('V1-Pou6f2', 'V3'): ['Gnai2 -> Adcy7'], ('V1-Rensh', 'DI6'): ['Rgmb -> Bmpr1b'], ('V1-Rensh', 'MN'): ['Tgfb2 -> Acvr1_Tgfbr2', 'Lama3 -> Itga7_Itgb1', 'Lama1 -> Itga7_Itgb1', 'Rgmb -> Bmpr1b'], ('V1-Rensh', 'V0v'): ['Rgmb -> Bmpr1b'], ('V1-Rensh', 'V1-Pou6f2'): ['Rgmb -> Bmpr1b'], ('V1-Rensh', 'V2a-2'): ['Rgmb -> Bmpr1b'], ('V1-Rensh', 'V2b'): ['Rgmb -> Bmpr1b'], ('V1-Rensh', 'V3'): ['Rgmb -> Bmpr1b'], ('V1-Sp8', 'DI6'): ['Col6a1 -> Itga9_Itgb1'], ('V1-Sp8', 'MN'): ['Lama1 -> Itga7_Itgb1', 'Col6a1 -> Itga6', 'Col6a1 -> Itga9_Itgb1', 'Ntn1 -> Mcam'], ('V1-Sp8', 'V1-Foxp2'): ['Col6a1 -> Itga9_Itgb1'], ('V1-Sp8', 'V1-Pou6f2'): ['Col6a1 -> Itga9_Itgb1'], ('V1-Sp8', 'V1-Rensh'): ['Col6a1 -> Itga6'], ('V1-Sp8', 'V2a-1'): ['Col6a1 -> Itga9_Itgb1'], ('V1-Sp8', 'V2a-2'): ['Col6a1 -> Itga9_Itgb1'], ('V1-Sp8', 'V3'): ['Col6a1 -> Itga6', 'Col6a1 -> Itga9_Itgb1'], ('V2a-1', 'DI6'): ['Apoa1 -> Lrp2', 'Apoa1 -> Ldlr', 'Hspa8 -> Lrp2', 'Fam3c -> Lifr', 'P4hb -> Mttp'], ('V2a-1', 'MN'): ['Qdpr -> Dysf', 'Apoa1 -> Ldlr', 'Fam3c -> Lifr', 'Apoa1 -> Abca1', 'Vegfb -> Ret'], ('V2a-1', 'V0d'): ['Apoa1 -> Ldlr', 'Arpc5 -> Ldlr', 'Agrp -> Sdc3'], ('V2a-1', 'V0v'): ['Ado -> Adora1', 'Qdpr -> Dysf', 'P4hb -> Mttp', 'Cthrc1 -> Fzd3', 'Cthrc1 -> Ror2'], ('V2a-1', 'V1-Foxp2'): ['Apoa1 -> Ldlr', 'P4hb -> Mttp', 'Penk -> Oprd1', 'Qdpr -> Dysf', 'Vegfb -> Ret'], ('V2a-1', 'V1-Pou6f2'): ['Qdpr -> Dysf', 'P4hb -> Mttp', 'Penk -> Oprk1', 'Fam3c -> Lifr', 'Cthrc1 -> Fzd3'], ('V2a-1', 'V1-Rensh'): ['Efna2 -> Epha8', 'Fam3c -> Lifr', 'Enho -> Gpr19', 'Bsg -> Slc16a1', 'Efna3 -> Epha8'], ('V2a-1', 'V1-Sp8'): ['Vegfb -> Ret', 'Cthrc1 -> Fzd3'], ('V2a-1', 'V2a-1'): ['Apoa1 -> Ldlr', 'Agrp -> Sdc3', 'Fam3c -> Lamp1', 'Ado -> Adora1', 'Arpc5 -> Ldlr'], ('V2a-1', 'V2a-2'): ['P4hb -> Mttp', 'Cthrc1 -> Fzd3', 'Penk -> Oprd1', 'Fam3c -> Lamp1', 'Agrp -> Sdc3'], ('V2a-1', 'V2b'): ['Apoa1 -> Lrp2', 'Arpc5 -> Lrp2', 'Hspa8 -> Lrp2'], ('V2a-1', 'V3'): ['Apoa1 -> Ldlr', 'Omg -> Rtn4rl1', 'Qdpr -> Dysf', 'Vegfb -> Ret', 'Gnai2 -> Adcy7'], ('V2a-2', 'DI6'): ['Hspa8 -> Lrp2'], ('V2a-2', 'MN'): ['Tgfb2 -> Acvr1_Tgfbr2', 'Lin7c -> Abca1', 'Fabp5 -> Rxra', 'Dlk1 -> Notch2'], ('V2a-2', 'V0d'): ['Agrp -> Sdc3'], ('V2a-2', 'V1-Rensh'): ['Efna3 -> Epha8', 'Bsg -> Slc16a1'], ('V2a-2', 'V2a-1'): ['Agrp -> Sdc3'], ('V2a-2', 'V2a-2'): ['Agrp -> Sdc3'], ('V2a-2', 'V2b'): ['Hspa8 -> Lrp2'], ('V3', 'MN'): ['Cgn -> Tgfbr2', 'Lama3 -> Itga7_Itgb1'], ('V3', 'V1-Rensh'): ['Cgn -> Tgfbr1'], ('V3', 'V3'): ['Cgn -> Tgfbr1', 'Gnai2 -> Adcy7']}
{('V0d', 'MN'): ['Lama2 -> Itga7_Itgb1'], ('V2a-1', 'MN'): ['Apoa1 -> Abca1', 'Dll3 -> Notch2'], ('V2a-1', 'V0v'): ['Cthrc1 -> Ror2'], ('V2a-1', 'V1-Rensh'): ['Efna2 -> Epha8', 'Enho -> Gpr19'], ('V2a-1', 'V2a-1'): ['Nptx1 -> Nptxr', 'S100a10 -> Htr1b'], ('V2a-1', 'V3'): ['Omg -> Rtn4rl1'], ('V2a-2', 'MN'): ['Dlk1 -> Notch2'], ('V3', 'MN'): ['Cgn -> Tgfbr2']}

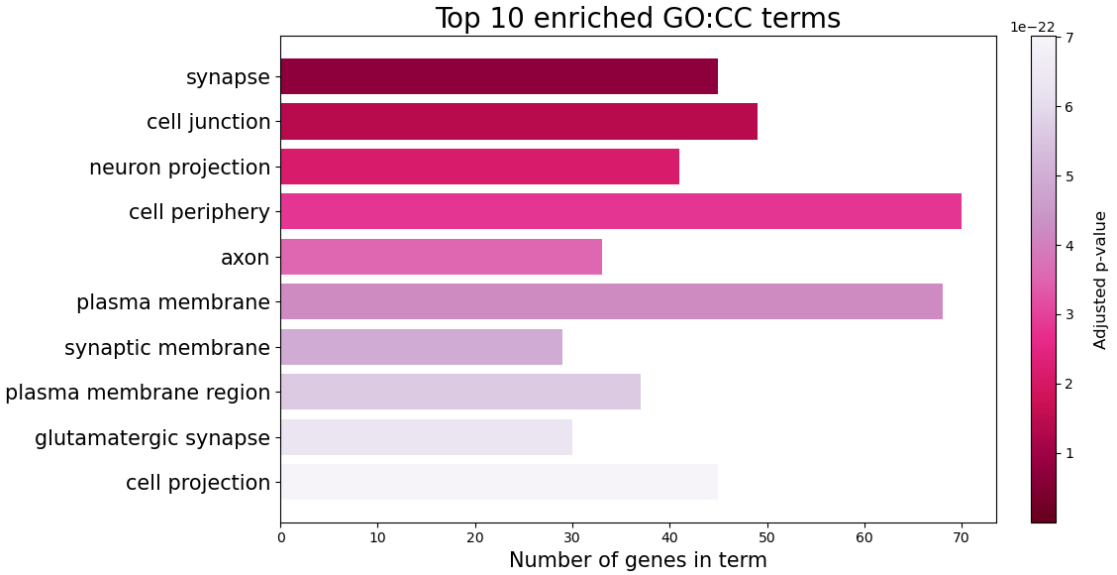
[186]:
```
def top_interactions(df):
    top_interactions = df.copy()
    top_interactions = top_interactions[(top_interactions['cellphone_pvals'] <␣
 ↪0.05) & (top_interactions['magnitude_rank'] < 0.05)]
    top_interactions = top_interactions.sort_values('magnitude_rank',␣
 ↪ascending=True)

    ligands_of_interest = top_interactions['ligand_complex'].unique().tolist()
    receptors_of_interest = top_interactions['receptor_complex'].unique().
 ↪tolist()

    interacting_genes = [gene.upper() for gene in set(ligands_of_interest +␣
 ↪receptors_of_interest)]

    print(f"Number of top interactions: {len(top_interactions)}")
    print(f"Number of unique ligands: {len(ligands_of_interest)}")
    print(f"Number of unique receptors: {len(receptors_of_interest)}")
    print(f"Number of unique implicated genes: {len(interacting_genes)}")

    print("Implicated genes:", interacting_genes)
    save_path = '/Users/sabrina/Library/CloudStorage/
 ↪OneDrive-UniversityofCopenhagen/Thesis/Figures/Final/top_interactions.txt'
    with open(save_path, 'w') as f:
        for gene in interacting_genes:
            f.write(f"{gene}\n")

top_interactions(adata.uns['liana_res'])
```

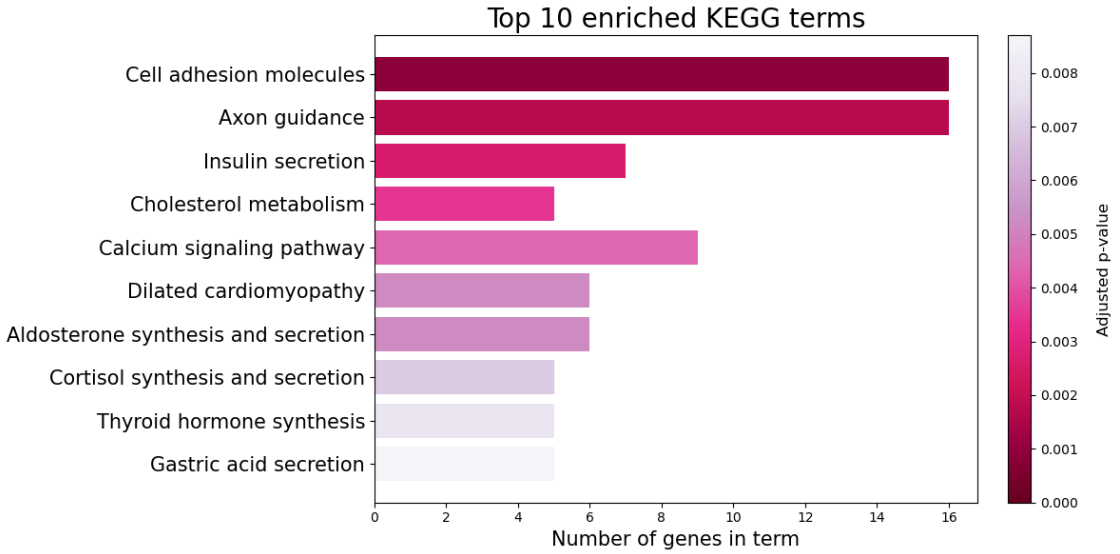
```
Number of top interactions: 2119
Number of unique ligands: 42
Number of unique receptors: 50
Number of unique implicated genes: 86
Implicated genes: ['TMEM67', 'LRP2', 'ALCAM', 'PTPRG', 'ARF1', 'ROBO2', 'NLGN2',
'CHL1', 'NRCAM', 'LDLR', 'EPHA6', 'DSCAM', 'F8', 'CNTN5', 'NCAM2', 'WNT5B',
'ADCY9', 'FZD3_LRP6', 'APOA1', 'ITGA3_ITGB1', 'GFRA1', 'CNTN1', 'IL1RAPL1',
'ROBO1', 'NRXN2', 'CNTN6', 'CADM1', 'HTR2C', 'CNTN4', 'LAMA1', 'CADM3', 'EPHA5',
'AGRN', 'RPSA', 'CGN', 'LRPAP1', 'FGFR1', 'FSTL5', 'NXPH1', 'DCC', 'TGFBR2',
'CACNA1C', 'LRRC4C', 'ITGA6', 'NRG2', 'EFNA5', 'GRM7', 'FGF14', 'LGI2',
'SEMA6A', 'COL6A1', 'NTNG1', 'APLP1', 'NRP1_PLXNA3', 'APP', 'NRXN1', 'SEMA4A',
'TXLNA', 'EPHB1', 'NCAM1', 'APLP2', 'ADCY8', 'RIMS1', 'SDK2', 'NRP2', 'TCTN1',
'GNAS', 'NLGN3', 'CHRM3', 'ADAM23', 'STX1A', 'LRP1', 'CNTN3', 'PTPRS', 'NRP1',
'ATP1A3', 'ADAM22', 'PLXNA4', 'NLGN1', 'NTRK3', 'EPHA10', 'SLIT2', 'FGFR2',
'SLIT3', 'FGF12', 'LIN7C']
```

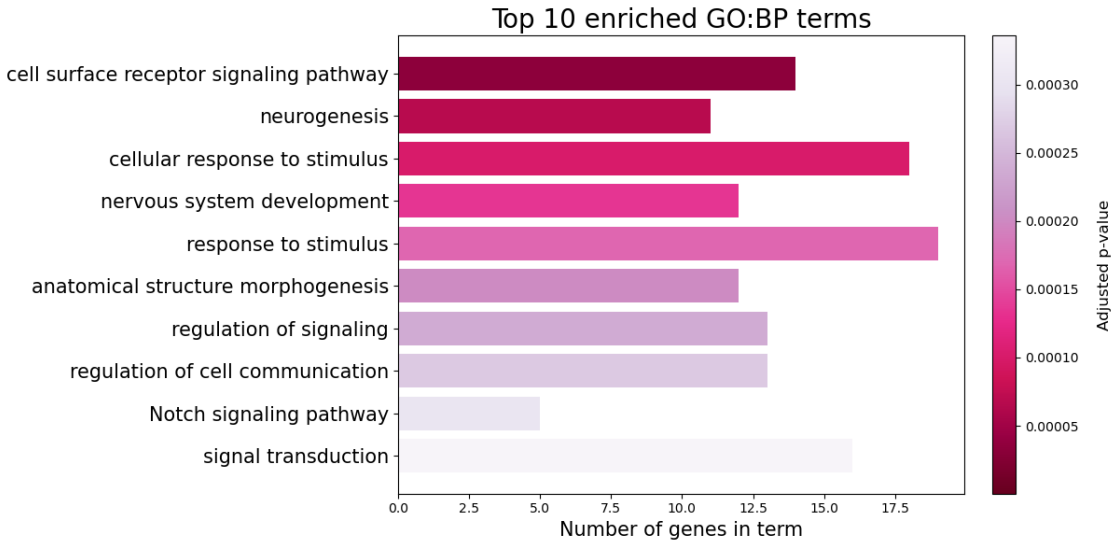
[187]:
```
def plot_gprofiler_results(file, name):
    df = pd.read_csv(file)
```

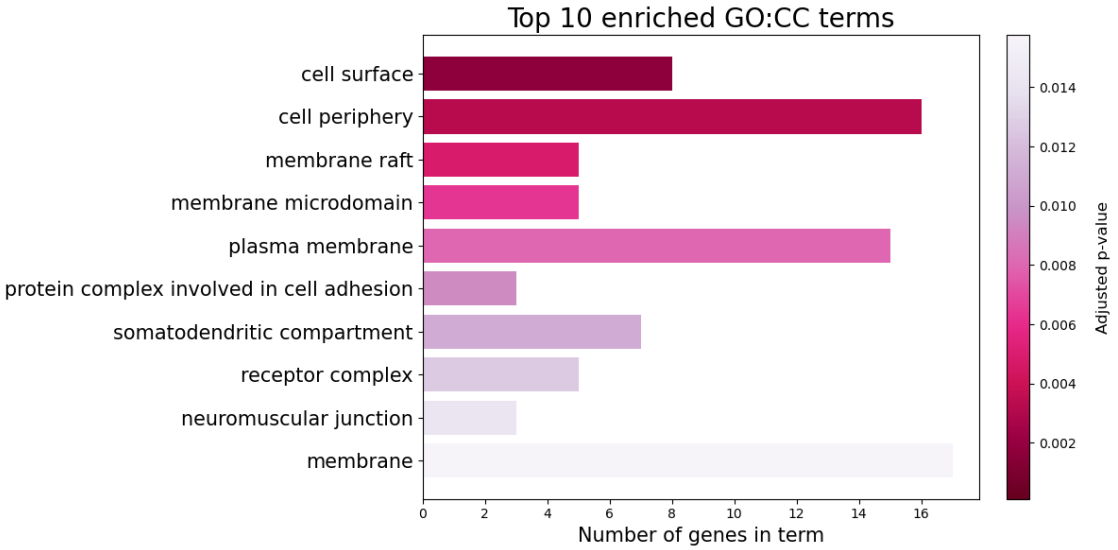
```
    for source in ['GO:BP', 'GO:CC', 'KEGG']:
        df_source = df[df['source'] == source].copy()
        df_top = df_source.sort_values('negative_log10_of_adjusted_p_value',␣
↪ascending=False).head(10)

        fig = plt.figure(figsize=(12, 6))
        bars = plt.barh(
            y=df_top['term_name'],
            width=df_top['intersection_size'],
            color=plt.cm.PuRd_r(df_top['adjusted_p_value'].rank(method='min') /␣
↪len(df_top)))
        sm = plt.cm.ScalarMappable(cmap='PuRd_r', norm=plt.
↪Normalize(vmin=df_top['adjusted_p_value'].min(),␣
↪vmax=df_top['adjusted_p_value'].max()))
        sm.set_array([])

        plt.xlabel('Number of genes in term', fontsize=15)
        plt.tick_params(axis='y', left=True, labelsize=15)
        plt.title(f'Top 10 enriched {source} terms', fontsize=20)
        plt.gca().invert_yaxis()
        cbar = plt.colorbar(sm, ax=plt.gca(), orientation='vertical', pad=0.04)
        cbar.set_label('Adjusted p-value', rotation=90, labelpad=15,␣
↪fontsize=12)
        plt.tight_layout()
        plt.show()
        fig.savefig(f'/Users/sabrina/Library/CloudStorage/
↪OneDrive-UniversityofCopenhagen/Thesis/Figures/{name}_{source}.svg',␣
↪format='svg', dpi=300, bbox_inches='tight')

plot_gprofiler_results("../gprofiler_result.csv", "result")
plot_gprofiler_results("../gprofiler_only1.csv", "only1")
plot_gprofiler_results("../gprofiler_5percent.csv", "5percent")
```

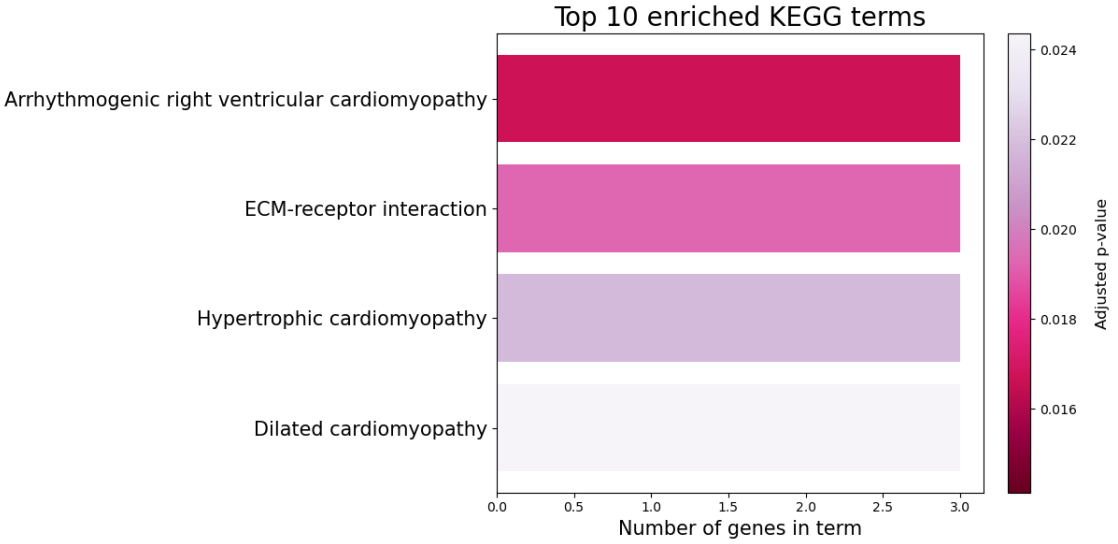

Top 10 enriched GO:BP terms
neuron projection development
nervous system development
neurogenesis
neuron development
neuron differentiation
axon development
generation of neurons
plasma membrane bounded cell projection organization
axonogenesis
cell projection organization
0
10
20
30
40
50
60
Number of genes in term
1e−32
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
Adjusted p-value

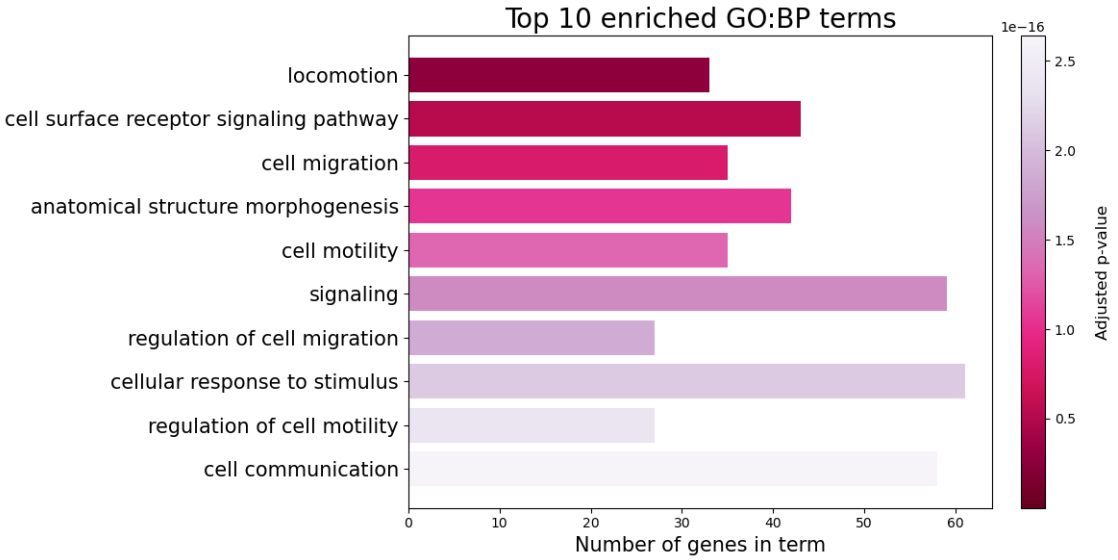

Top 10 enriched GO:CC terms
synapse
cell junction
neuron projection
cell periphery
axon
plasma membrane
synaptic membrane
plasma membrane region
glutamatergic synapse
cell projection
0
10
20
30
40
50
60
70
Number of genes in term
1e−22
7
6
5
4
3
2
1
Adjusted p-value

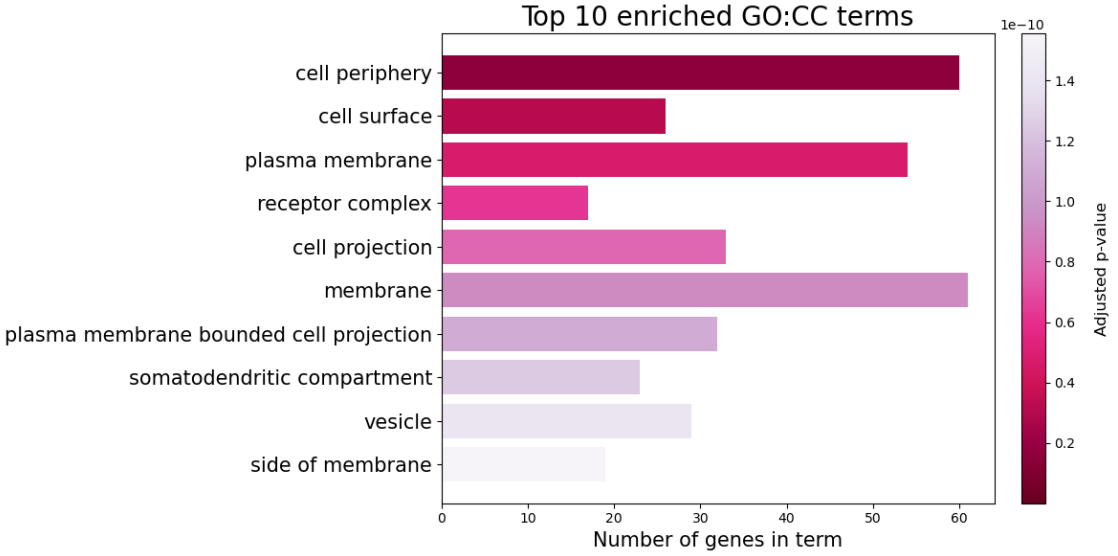

Top 10 enriched KEGG terms
Cell adhesion molecules
Axon guidance
Insulin secretion
Cholesterol metabolism
Calcium signaling pathway
Dilated cardiomyopathy
Aldosterone synthesis and secretion
Cortisol synthesis and secretion
Thyroid hormone synthesis
Gastric acid secretion
0
2
4
6
8
10
12
14
16
Number of genes in term
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
Adjusted p-value

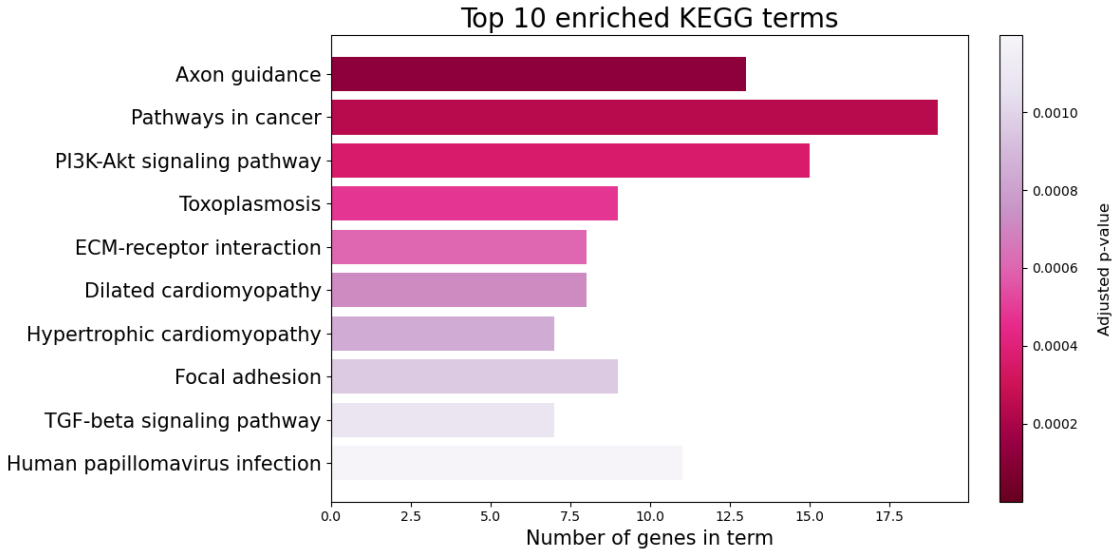

Top 10 enriched GO:BP terms
cell surface receptor signaling pathway
neurogenesis
cellular response to stimulus
nervous system development
response to stimulus
anatomical structure morphogenesis
regulation of signaling
regulation of cell communication
Notch signaling pathway
signal transduction
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Number of genes in term
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
Adjusted p-value


Top 10 enriched GO:CC terms
cell surface
cell periphery
membrane raft
membrane microdomain
plasma membrane
protein complex involved in cell adhesion
somatodendritic compartment
receptor complex
neuromuscular junction
membrane
0
2
4
6
8
10
12
14
16
Number of genes in term
0.014
0.012
0.010
0.008
0.006
0.004
0.002
Adjusted p-value


Top 10 enriched KEGG terms
Arrhythmogenic right ventricular cardiomyopathy
ECM-receptor interaction
Hypertrophic cardiomyopathy
Dilated cardiomyopathy
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Number of genes in term
0.024
0.022
0.020
0.018
0.016
Adjusted p-value


Top 10 enriched GO:BP terms
locomotion
cell surface receptor signaling pathway
cell migration
anatomical structure morphogenesis
cell motility
signaling
regulation of cell migration
cellular response to stimulus
regulation of cell motility
cell communication
0
10
20
30
40
50
60
Number of genes in term
1e−16
2.5
2.0
1.5
1.0
0.5
Adjusted p-value


Top 10 enriched GO:CC terms
cell periphery
cell surface
plasma membrane
receptor complex
cell projection
membrane
plasma membrane bounded cell projection
somatodendritic compartment
vesicle
side of membrane
0
10
20
30
40
50
60
Number of genes in term
1e−10
1.4
1.2
1.0
0.8
0.6
0.4
0.2
Adjusted p-value



6 Extra functions

[ ]:
```
def plot_violin(freq_matrices, method_key):

    """
    Create a plot of N methods used to infer cell-cell communication subplots␣
↪with violin and box plots.
    """
    key, method, mag_col, spec_col = method_key

    fig, axs = plt.subplots(1, 2, figsize=(6, 6))
    fig.subplots_adjust(wspace=0.5)
    axs = axs.flatten()
    for i, measure in enumerate(list(freq_matrices.keys())):
        data = freq_matrices[measure].values.flatten()
        ax = axs[i]
        plot_df = pd.DataFrame({'scores': data})
        sns.violinplot(y='scores', data=plot_df, color="#eaa5be", linewidth=0,␣
↪ax=ax)
        sns.boxplot(y='scores', data=plot_df, width=0.3, boxprops=dict(alpha=0.
↪7), showcaps=True, showfliers=True, color='#cc4778', ax=ax)
        ax.set_title(measure)
        ax.set_ylabel(f'{measure}')
        if i >= 4:
            pos = ax.get_position()
            axs[i].set_position([
                pos.x0 + 0.12,
```

```
                pos.y0,
                pos.width,
                pos.height
            ])
    plt.suptitle(f"Distribution of cell-cell communication scores␣
↪({measure})\n{method}", fontsize=12, y=1.01)
    plt.show()

def plot_survey_pairs_violin(freq_mat, method_key, data_name, survey_pairs=pd.
↪read_csv('../survey_pairs.csv', index_col=0).T):
    """
    Plots the distribution of communication scores, considering only known pairs
    defined in the `survey_pairs` DataFrame.
    """
    label, method, mag_col, spec_col = method_key
    plot_data = []

    if all(survey_pairs.index == freq_mat.index) and all(survey_pairs.columns␣
↪== freq_mat.columns):
        ev_mat = freq_mat[survey_pairs == "evidence"].values.flatten()
        ev_mat = ev_mat[~np.isnan(ev_mat)]
        unknown_mat = freq_mat[survey_pairs == "unknown"].values.flatten()
        unknown_mat = unknown_mat[~np.isnan(unknown_mat)]
        sup_mat = freq_mat[survey_pairs == "support"].values.flatten()
        sup_mat = sup_mat[~np.isnan(sup_mat)]
        pred_mat = freq_mat[survey_pairs == "predicted"].values.flatten()
        pred_mat = pred_mat[~np.isnan(pred_mat)]
        low_mat = freq_mat[survey_pairs == "low/absent"].values.flatten()
        low_mat = low_mat[~np.isnan(low_mat)]
    else:
            print("Error: survey_pairs does not match freq_mat order.")

    plot_data.extend([pd.DataFrame({"value": mat, "evidence": f"CT pairs {i}"})␣
↪for i, mat in enumerate([ev_mat, unknown_mat, sup_mat, pred_mat, low_mat])])

    plot_df = pd.concat(plot_data, ignore_index=True)

    fig, ax = plt.subplots(figsize=(8, 6))
    sns.violinplot(data=plot_df, x="evidence", y="value", inner="box",␣
↪color="#d6a8e3", linewidth=1,
                   density_norm="width", ax=ax)
    ax.set_xticklabels(ax.get_xticklabels(), fontsize=10)
    sns.boxplot(data=plot_df, x="evidence", y="value", width=0.2,␣
↪boxprops=dict(alpha=0.7), showcaps=True,
                showfliers=True, color="#bf52c8", ax=ax)
    ax.set_title(f"Distribution of frequency of LR pairs by", fontsize=16)
```

```
    ax.set_xlabel(f"Cell Type Pairs")
    ax.set_ylabel(f"Frequency score")
    plt.suptitle(f'{data_name}', fontsize=10, weight='bold', y=1.005)
    plt.tight_layout(w_pad=15)
    plt.show()

def top3_byCTpair(adata, freq_mat):
    final_df = pd.DataFrame(columns=freq_mat.columns.unique(), index=freq_mat.
 ↪index.unique())
    top_interactions = adata.uns['liana_res'].copy()
    top_interactions = top_interactions[(top_interactions['cellphone_pvals'] <␣
 ↪0.05) & (top_interactions['magnitude_rank'] < 0.05)]
    top_interactions = top_interactions.sort_values('magnitude_rank',␣
 ↪ascending=True)
    for source in freq_mat.columns.unique():
        for target in freq_mat.index.unique():
            top_genes = []
            ligands_of_interest = []
            receptors_of_interest = []
            interacting_genes = []
            top_genes = top_interactions[(top_interactions['source'] == source)␣
 ↪& (top_interactions['target'] == target)]
            ligands_of_interest = top_genes['ligand_complex'].unique().tolist()
            receptors_of_interest = top_genes['receptor_complex'].unique().
 ↪tolist()
            interacting_genes = [gene.upper() for gene in␣
 ↪set(ligands_of_interest + receptors_of_interest)]
            print(f"Top 3 genes for {source} -> {target}:")
            print(interacting_genes[:3])
            final_df[source][target] = interacting_genes[:3]

top3_byCTpair(adata, freq_matrices['magnitude']['Liana+␣
 ↪Consensus']["Frequency"])
```