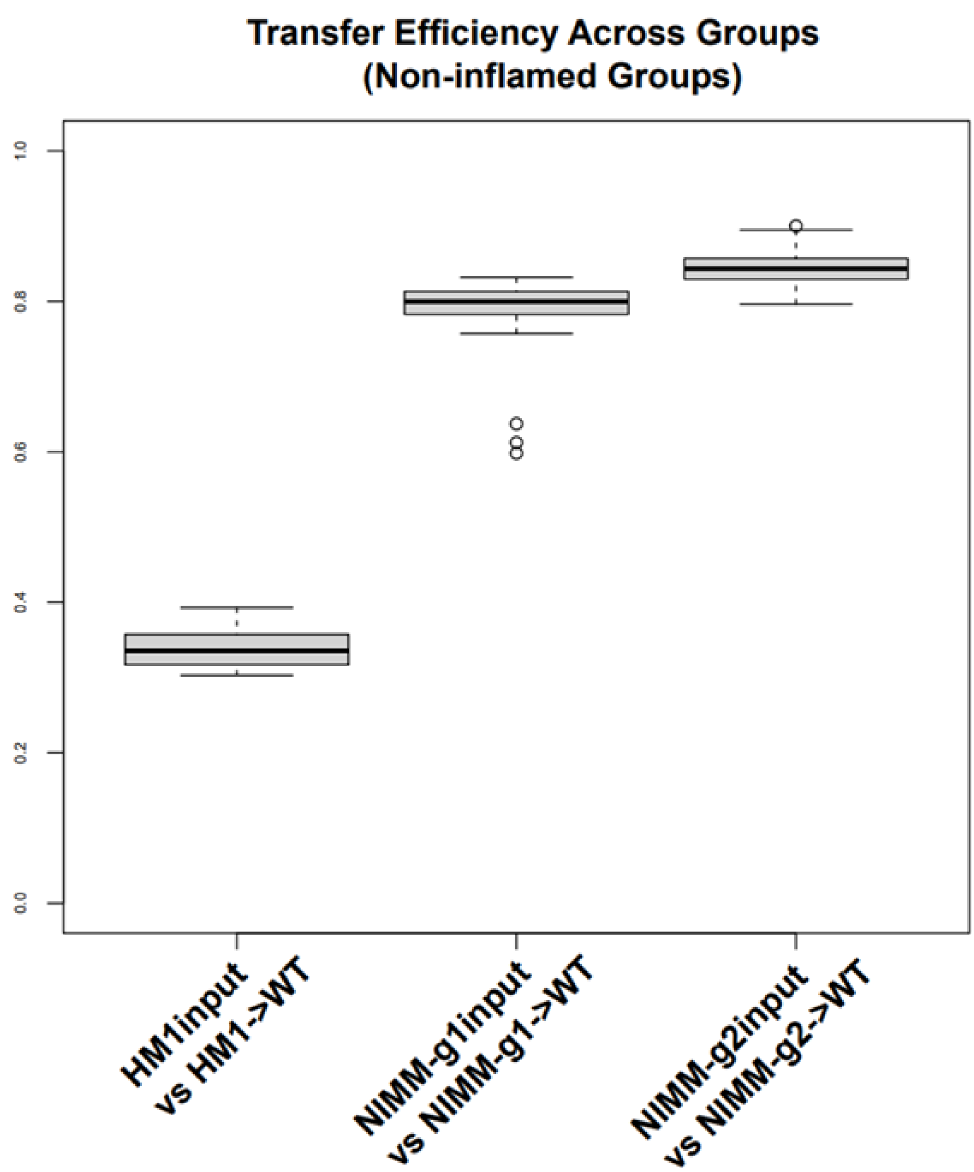
Contents

- **Aim**
- **Main Python Methods**
- **Data Processing Steps**
  - Step 1: Retrieve original counts table
  - Step 2: Retrieve original metadata table
  - Step 3: Create dictionary from metadata table
  - Step 4: Calculate total read counts per sample
  - Step 5: Extract counts for all six sample groups
  - Step 6: Normalize count values for all six sample groups
  - Step 7: Calculate Spearman Correlation Coefficients for each sample group
  - Step 8: Generate final figure
  - Step 9: Compare original and regenerated figures

Aim: Replicate Figure 4B



Main Python Methods

In [20]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import re
from scipy import stats
from scipy.special import comb

# ******************************* MAIN METHODS *******************************
# calculate Pearson correlation coefficients from input set of samples
def get_inter_corr_values(df1, df2):
    corr_values = [stats.pearsonr(df1.loc[:, col_A], df2.loc[:, col_B]).statis
tic \
            for col_A in df1.columns for col_B in df2.columns]
    # Remove nan values
    corr_values = [val for val in corr_values if not np.isnan(val)]

    return corr_values

# Retrieve tab delimited file
def read_csv_file(file_path, skiprows=None, header = 0, sep = '\t', index_col=
False):
    df = pd.read_csv(file_path, skiprows=skiprows, sep=sep, header=header, ind
ex_col=index_col)

    return df

# Create dictionary from metadata table
def get_dict_from_metadata(input_df):
    mydict = {}
    for row in input_df.iterrows():
        obj = row[1]
        sample_id = obj['SampleID']
        key = obj['FMTGroupFMTsourcegtRecipientbackground']
        if key not in mydict:
            mydict[key] = [sample_id]
        else:
            mydict[key].append(sample_id)
    return mydict

# ******************************* HELPER METHODS *****************************
**
# Normalize sample counts
def _get_norm_counts(input_df, ser_sample_count_sums):
    overall_mean_count = ser_sample_count_sums.mean()
    df_norm_logged =    pd.DataFrame()
    for col_name in input_df.columns:
        df_norm_logged.loc[:, col_name] = \
        np.log10(((input_df.loc[:, col_name] / ser_sample_count_sums[col_name
]) * overall_mean_count) + 1)

    return df_norm_logged

# Get tuples of sample column names
def _get_tup_list(col_names):
    tup_list = []
    corr_result = []
    col_name_length = len(col_names)
    for i in range(col_name_length):
```

```
        for j in range(i+1, col_name_length):
            tup_list.append((i, j))

    return tup_list

# ****************************** INPUT FILE PATHS ***************************
****
# Get current working directory
current_working_dir = os.getcwd()
# original counts table
asv_tbl_file_path = os.path.join(current_working_dir, 'asv_biom-with-taxonomy.
txt')
# original metadata table
metadata_file_path = os.path.join(current_working_dir, 'mappingMetadata.txt')
```

Data Processing Steps

Step 1: Retrieve original counts table

In [21]:

```
df_asv = read_csv_file(asv_tbl_file_path, 1)
df_asv = df_asv.astype({col:'int32' for col in df_asv.columns[1:-1] }, copy=Fa
lse)
df_asv
```

Out[21]:

	#OTU ID	1gKO.1	1gKO.2	1gKO.3	1gWT.1	1gWT.2	1gWT.3	2gKO.1
0	1ba8c796d07406783c96d016a6a5cace	13615	16637	17148	20227	23630	25656	14832
1	a6c38249aff7768283faf6cfbdeb05a8	26439	30129	19743	8955	10759	7074	18489
2	062f38ff92cfaee0654200b6f5be5ddf	7451	8774	8754	174	214	148	21958
3	1183cc23f552d81e63c93ca9fcba2f2c	225	223	184	13762	16856	18692	269
4	5e15ecfb579e72bf87c0bea3920bbf42	10108	12117	8633	10027	11910	7424	5979
...	...	...	...	...	...	...	...	...
4070	92bb8f4683ef5c8651e7d34dbb37ab2e	0	0	0	0	0	0	0
4071	92f09070a4fd5786bb34e756217e6ee1	0	0	0	0	0	0	0
4072	919b82324c41ed0046323c63aa1550da	0	0	0	0	0	0	0
4073	dbc0dad15ec1c8ad9d826cab94e18696	0	0	0	0	0	0	0
4074	1ff2d07d10264c23dc43e08d3097cd7c	0	0	0	0	0	0	0

4075 rows × 112 columns

Step 2: Retrieve original metadata table

In [22]:
```
df_metadata = read_csv_file(metadata_file_path)
df_metadata
```

Out[22]:

	SampleID	UniversalCageNumber	Background	FMTGroupFMTsourcegtRecipientbackground	Passage
0	F8-1	F8-cage-1	129.IL10KO	1gKOgtKO	8
1	F8-2	F8-cage-1	129.IL10KO	1gKOgtKO	8
2	F8-3	F8-cage-2	129.IL10KO	1gKOgtKO	8
3	F8-4	F8-cage-2	129.IL10KO	1gKOgtKO	8
4	F8-5	F8-cage-3	129.IL10KO	1gKOgtKO	8
...	...	...	...	...	...
105	1gWT.2	NaN	NaN	1gWTinput	1gWT
106	1gWT.3	NaN	NaN	1gWTinput	1gWT
107	2gWT.1	NaN	NaN	2gWTinput	2gWT
108	2gWT.2	NaN	NaN	2gWTinput	2gWT
109	2gWT.3	NaN	NaN	2gWTinput	2gWT

110 rows × 8 columns

Step 3: Create dictionary from metadata table

In [4]:
```
dict_metadata = get_dict_from_metadata(df_metadata)
dict_metadata.keys()
```

Out[4]:
```
dict_keys(['1gKOgtKO', '2gKOgtKO', '1gWTgtKO', '1gWTgtWT', '2gWTgtWT', 'hFM
T.1.2.3.gtKO', 'hFMT.3.4.5.gtKO', 'hFMT.1.2.3.gtWT', 'hFMT.1.2.3.input', 'hF
MT.3.4.5.input', '1gKOinput', '2gKOinput', '1gWTinput', '2gWTinput'])
```

Step 4: Calculate total read counts per sample

In [5]:
```
sample_count_sums = df_asv.iloc[:, 1:-1].sum(axis=0)
sample_count_sums
```

Out[5]:
```
1gKO.1       118256
1gKO.2       141891
1gKO.3       123292
1gWT.1       119717
1gWT.2       146158
              ...
h1-2-3.2     135326
h1-2-3.3     133745
h3-4-5.1     129613
h3-4-5.2     140316
h3-4-5.3     132984
Length: 110, dtype: int64
```

Step 5: Extract counts for target sample groups

In [6]:
```
# 'hFMT.1.2.3.input' -->  'HM1Input'
key_name ='hFMT.1.2.3.input'
print('Grouped columns: {}'.format(dict_metadata[key_name]))
HM1_Input = df_asv[[col_name for col_name in dict_metadata[key_name] ]]
HM1_Input
```

```
Grouped columns: ['h1-2-3.1', 'h1-2-3.2', 'h1-2-3.3']
```

Out[6]:

	h1-2-3.1	h1-2-3.2	h1-2-3.3
0	1483	1515	1402
1	1129	691	725
2	2927	3523	3254
3	298	231	238
4	218	160	111
...	...	...	...
4070	0	0	0
4071	0	0	0
4072	0	0	0
4073	0	0	0
4074	0	0	0

4075 rows × 3 columns

In [7]:
```
# 'hFMT.1.2.3.gtWT' -->  'HM1->WT'
key_name = 'hFMT.1.2.3.gtWT'
print('Grouped columns: {}'.format(dict_metadata[key_name]))
print('Number of expected correlation values: {}'.format(int(comb(len(dict_met
adata[key_name]), 2))))
HM1_WT = df_asv[[col_name for col_name in dict_metadata[key_name] ]]
HM1_WT
```

```
Grouped columns: ['F1-16', 'F1-17', 'F1-18', 'F1-19', 'F1-20', 'F1-21', 'F1-
22']
Number of expected correlation values: 21
```

Out[7]:

	F1-16	F1-17	F1-18	F1-19	F1-20	F1-21	F1-22
0	19186	10727	29599	32588	32950	18853	5199
1	2938	6816	3233	1118	1710	1047	2202
2	9072	10348	5289	176	196	207	269
3	247	258	275	18297	22544	19687	21014
4	2389	3858	1904	3904	2928	5286	4884
...	...	...	...	...	...	...	...
4070	0	0	0	0	0	0	0
4071	0	0	0	0	0	0	0
4072	0	0	0	0	0	0	0

4072	0	0	0	0	0	0	0
4073	0	0	0	0	0	0	0
4074	0	0	0	0	0	0	0

4075 rows × 7 columns

In [8]:

```
# '1gWTinput' -->  'NIMM-g1input'
key_name ='1gWTinput'
print('Grouped columns: {}'.format(dict_metadata[key_name]))
NIMM_g1input = df_asv[[col_name for col_name in dict_metadata[key_name] ]]
NIMM_g1input
```

```
Grouped columns: ['1gWT.1', '1gWT.2', '1gWT.3']
```

Out[8]:

	1gWT.1	1gWT.2	1gWT.3
0	20227	23630	25656
1	8955	10759	7074
2	174	214	148
3	13762	16856	18692
4	10027	11910	7424
...	...	...	...
4070	0	0	0
4071	0	0	0
4072	0	0	0
4073	0	0	0
4074	0	0	0

4075 rows × 3 columns

In [9]:

```
# '1gWTgtWT' -->  'NIMM-g1->WT'
key_name ='1gWTgtWT'
print('Grouped columns: {}'.format(dict_metadata[key_name]))
NIMM_g1_WT = df_asv[[col_name for col_name in dict_metadata[key_name] ]]
NIMM_g1_WT
```

```
Grouped columns: ['F8-27', 'F8-28', 'F8-29', 'F8-30', 'F8-31', 'F8-32', 'F8-
33', 'F11-1', 'F11-2', 'F11-3', 'F11-4', 'F11-5', 'F11-6', 'F11-7', 'F11-8',
'F11-9', 'F11-10', 'F11-11']
```

Out[9]:

	F8-27	F8-28	F8-29	F8-30	F8-31	F8-32	F8-33	F11-1	F11-2	F11-3	F11-4	F11-5	F11-6	F
0	21313	6113	24573	19347	18028	10482	11761	34742	24360	31173	45582	18929	22349	1
1	300	972	397	332	573	825	1995	410	434	376	376	370	337	
2	135	243	197	193	99	207	237	129	291	167	282	412	190	
3	21071	16272	27247	21302	26135	23762	22465	25848	26231	19490	22181	18524	12915	2
4	2245	5227	5760	1189	2327	14279	7911	1416	2031	1899	2725	2679	1920	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4070	0	0	0	0	0	0	0	0	0	0	0	0	0	
4071	0	0	0	0	0	0	0	0	0	0	0	0	0	

4071	0	0	0	0	0	0	0	0	0	0	0	0	0
4072	0	0	0	0	0	0	0	0	0	0	0	0	0
4073	0	0	0	0	0	0	0	0	0	0	0	0	0
4074	0	0	0	0	0	0	0	0	0	0	0	0	0

4075 rows × 18 columns

In [10]:

```
# '2gWTinput' -->  'NIMM-g2input'
key_name ='2gWTinput'
print('Grouped columns: {}'.format(dict_metadata[key_name]))
NIMM_g2input = df_asv[[col_name for col_name in dict_metadata[key_name] ]]
NIMM_g2input
```

```
Grouped columns: ['2gWT.1', '2gWT.2', '2gWT.3']
```

Out[10]:

	2gWT.1	2gWT.2	2gWT.3
0	26154	27846	29545
1	487	298	403
2	221	152	77
3	18419	20399	21089
4	2446	1550	1249
...	...	...	...
4070	0	0	0
4071	0	0	0
4072	0	0	0
4073	0	0	0
4074	0	0	0

4075 rows × 3 columns

In [11]:

```
# '2gWTgtWT' -->  'NIMM-g2->WT'
key_name ='2gWTgtWT'
print('Grouped columns: {}'.format(dict_metadata[key_name]))
NIMM_g2_WT = df_asv[[col_name for col_name in dict_metadata[key_name] ]]
NIMM_g2_WT
```

```
Grouped columns: ['F8-34', 'F8-35', 'F8-36', 'F8-37', 'F8-38', 'F8-39', 'F8-
40', 'F8-41']
```

Out[11]:

	F8-34	F8-35	F8-36	F8-37	F8-38	F8-39	F8-40	F8-41
0	23089	23716	26435	13911	32356	18934	23958	18944
1	329	537	320	362	473	1848	258	282
2	172	254	238	146	225	135	189	66
3	11455	20177	11798	13910	13422	24886	20107	18889
4	4360	10659	3839	3613	5501	13735	2324	12215
...	...	...	...	...	...	...	...	...
4070	0	0	0	0	0	0	0	0
4071	0	0	0	0	0	0	0	0
4072	0	0	0	0	0	0	0	0

4072	0	0	0	0	0	0	0	0
4073	0	0	0	0	0	0	0	0
4074	0	0	0	0	0	0	0	0

4075 rows × 8 columns

Step 6: Normalize count values for target sample groups

In [12]:
```
HM1_Input_norm = _get_norm_counts(HM1_Input, sample_count_sums)
HM1_Input_norm
```

Out[12]:

	h1-2-3.1	h1-2-3.2	h1-2-3.3
0	3.170992	3.168281	3.139741
1	3.052636	2.827698	2.853624
2	3.466129	3.534613	3.505231
3	2.475230	2.353116	2.371106
4	2.340004	2.194478	2.041962
...	...	...	...
4070	0.000000	0.000000	0.000000
4071	0.000000	0.000000	0.000000
4072	0.000000	0.000000	0.000000
4073	0.000000	0.000000	0.000000
4074	0.000000	0.000000	0.000000

4075 rows × 3 columns

In [13]:
```
HM1_WT_norm = _get_norm_counts(HM1_WT, sample_count_sums)
HM1_WT_norm
```

Out[13]:

	F1-16	F1-17	F1-18	F1-19	F1-20	F1-21	F1-22
0	4.245267	3.999519	4.464795	4.552695	4.457225	4.238876	3.673620
1	3.430471	3.802595	3.503245	3.088422	3.172643	2.983868	3.300643
2	3.920013	3.983899	3.716960	2.287385	2.234151	2.281717	2.389137
3	2.356868	2.382421	2.434436	4.302027	4.292408	4.257674	4.280140
4	3.340672	3.555481	3.273402	3.631237	3.406096	3.686687	3.646482
...	...	...	...	...	...	...	...
4070	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
4071	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
4072	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
4073	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
4074	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

4075 rows × 7 columns

In [14]:
```
NIMM_g1input_norm = _get_norm_counts(NIMM_g1input, sample_count_sums)
NIMM_g1input_norm
```

Out[14]:

	1gWT.1	1gWT.2	1gWT.3
0	4.346750	4.327617	4.390865
1	3.992909	3.985949	3.831387
2	2.283615	2.286796	2.154970
3	4.179510	4.180916	4.253338
4	4.042010	4.030085	3.852357
...	...	...	...
4070	0.000000	0.000000	0.000000
4071	0.000000	0.000000	0.000000
4072	0.000000	0.000000	0.000000
4073	0.000000	0.000000	0.000000
4074	0.000000	0.000000	0.000000

4075 rows × 3 columns

In [15]:
```
NIMM_g1_WT_norm = _get_norm_counts(NIMM_g1_WT, sample_count_sums)
NIMM_g1_WT_norm
```

Out[15]:

	F8-27	F8-28	F8-29	F8-30	F8-31	F8-32	F8-33	F11-1	F11-2	F11
0	4.269712	3.868181	4.305071	4.313638	4.241945	4.043398	4.160521	4.485555	4.333532	4.4697
1	2.419820	3.069904	2.514711	2.549369	2.744909	2.939868	3.390166	2.558671	2.585453	2.5523
2	2.075047	2.468951	2.211733	2.314670	1.986115	2.340871	2.466282	2.059083	2.412411	2.2014
3	4.264752	4.293331	4.349929	4.355443	4.403212	4.398815	4.441569	4.357132	4.365668	4.2658
4	3.292482	3.800189	3.675105	3.102528	3.352961	4.177641	3.988323	3.096097	3.254786	3.2547
...	...	...	...	...	...	...	...	...	...	
4070	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
4071	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
4072	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
4073	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
4074	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000

4075 rows × 18 columns

In [16]:
```
NIMM_g2input_norm = _get_norm_counts(NIMM_g2input, sample_count_sums)
NIMM_g2input_norm
```

Out[16]:

	2gWT.1	2gWT.2	2gWT.3
0	4.410161	4.472699	4.474499
1	2.681041	2.503502	2.610372
2	2.338993	2.212436	1.896044

3	4.257896	4.337551	4.328077
4	3.381243	3.218516	3.100908
...	...	...	...
4070	0.000000	0.000000	0.000000
4071	0.000000	0.000000	0.000000
4072	0.000000	0.000000	0.000000
4073	0.000000	0.000000	0.000000
4074	0.000000	0.000000	0.000000

4075 rows × 3 columns

In [17]:

```
NIMM_g2_WT_norm = _get_norm_counts(NIMM_g2_WT, sample_count_sums)
NIMM_g2_WT_norm
```

Out[17]:

	F8-34	F8-35	F8-36	F8-37	F8-38	F8-39	F8-40	F8-41
0	4.326180	4.345537	4.473561	4.148775	4.388825	4.288827	4.446174	4.297041
1	2.481386	2.701315	2.557721	2.565278	2.554926	3.278493	2.479769	2.471266
2	2.201025	2.377136	2.429562	2.172667	2.233579	2.145017	2.345136	1.845337
3	4.021791	4.275356	4.123208	4.148744	4.006713	4.407534	4.370073	4.295778
4	3.602349	3.998236	3.635686	3.563373	3.619399	4.149421	3.433104	4.106475
...	...	...	...	...	...	...	...	...
4070	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
4071	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
4072	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
4073	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
4074	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

4075 rows × 8 columns

Step 7: Calculate Spearman Correlation Coefficients for each sample group

In [18]:

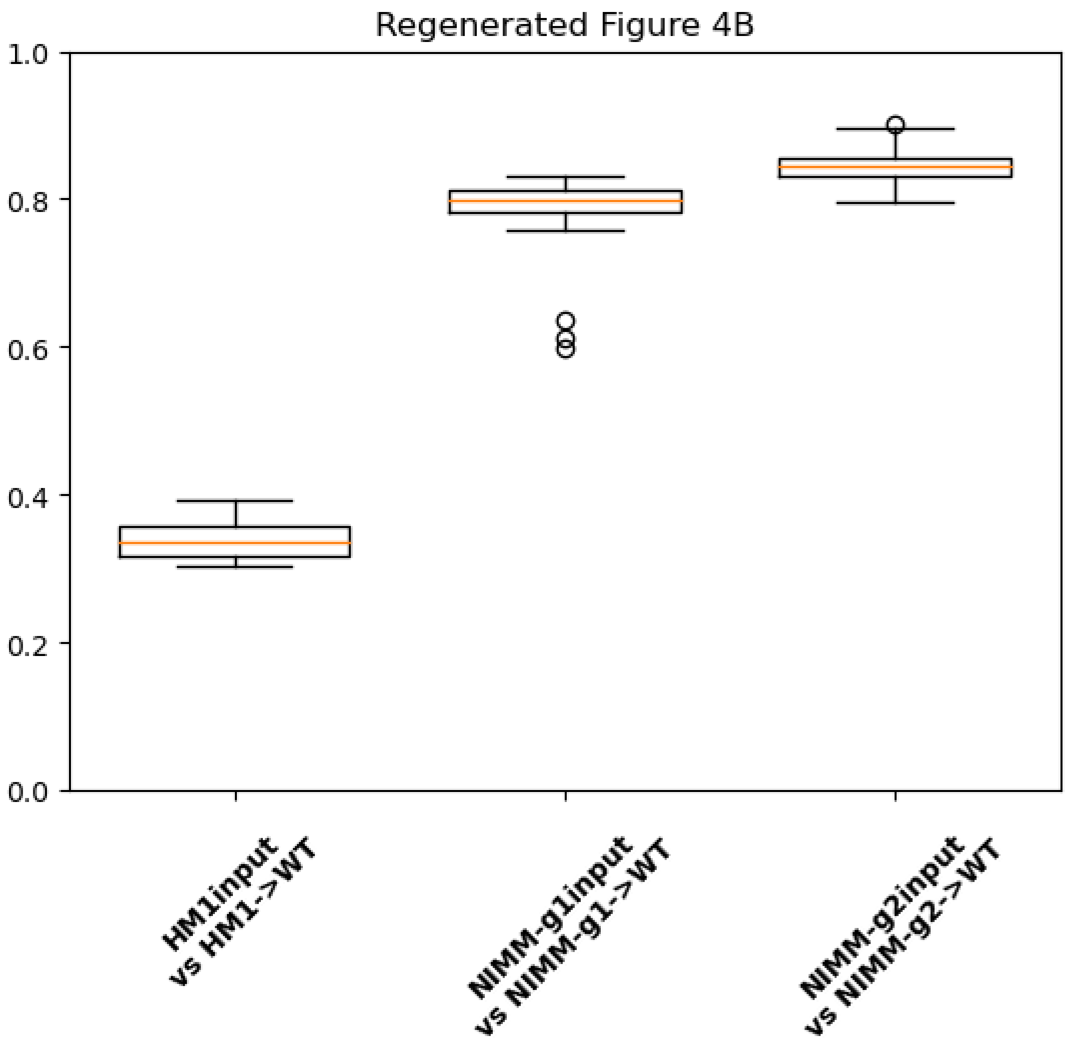
```
corr_value1 = get_inter_corr_values(HM1_Input_norm, HM1_WT_norm)
corr_value2 = get_inter_corr_values(NIMM_g1input_norm, NIMM_g1_WT_norm)
corr_value3 = get_inter_corr_values(NIMM_g2input_norm, NIMM_g2_WT_norm)
```

Step 8: Generate final figure

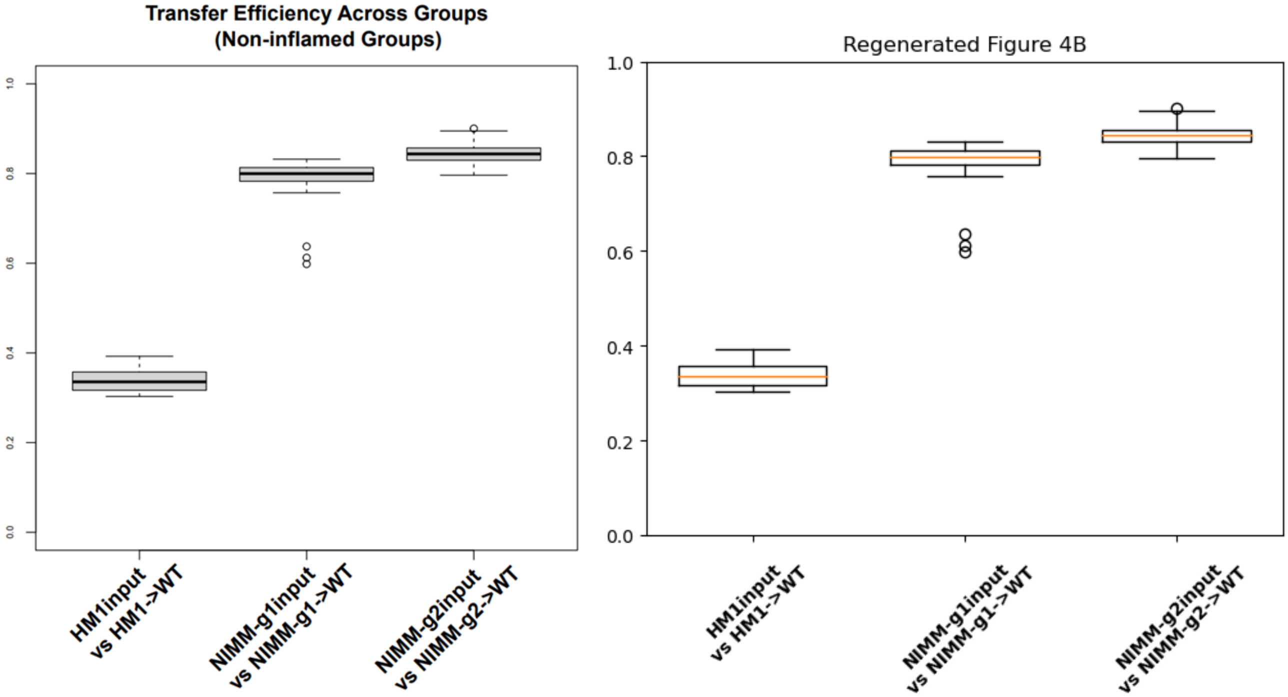
In [25]:

```
data = []
x_tick_labels = ['HM1input\nvs HM1->WT',
                 'NIMM-g1input\nvs NIMM-g1->WT',
                 'NIMM-g2input\nvs NIMM-g2->WT']

data.extend([corr_value1, corr_value2, corr_value3])
fig, ax = plt.subplots()
```

```
ax.boxplot(data, widths=0.7)
ax.set_title('Regenerated Figure 4B')
ax.set_yticks([0.0, 0.2, 0.4, 0.6, 0.8, 1.0])
ax.set_xticklabels(x_tick_labels,
                   rotation=45, fontsize=10, fontweight='bold')

plt.show()
```



Step 9: Compare original and regenerated figures


Transfer Efficiency Across Groups
(Non-inflamed Groups)
Regenerated Figure 4B
1.0
0.8
0.6
0.4
0.2
0.0
HM1input vs HM1->WT
NIMM-g1input vs NIMM-g1->WT
NIMM-g2input vs NIMM-g2->WT

```
In [ ]:
```