Weight Sum of Helpfulness Metrics

Here we are going to create a function that prints out the importance of each helpfulness metric. Once we have the function, it can be used to look at the overall weights as well as weights for individual workers.

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
```

In [13]:
```
df = pd.read_csv("../Data/all_metrics_upd.csv")
df = df[df['tone_similarity'] < 1]
df = df[df['coherence_score'] < 1]
df = df[df['pos_similarity'] < 1]
df = df[df['ai_coverage'] < 1]
df = df[df['ai_coverage'] > 0]
```

In [14]:
```
def calculate_metric_weights(df, worker_id=None):
    '''
    Calculate the weight of each helpfulness metric for a given writer.

    Input: csv file

    Output: dictionary of feature importance
    '''

    # Select relevant columns
    features = ['tone_similarity',      'pos_similarity',       'coherence_sco
    target = 'acceptance_status'

    # Filter for specific writer if worker_id is provided
    if worker_id:
        df = df[df['workerID'] == worker_id]

    # Drop rows with missing values
    df = df.dropna(subset=features + [target])

    # Convert target to binary, accepted is 1 and rejected is 0
    df[target] = df[target].apply(lambda x: 1 if x == 'accepted' else 0)

    if df[target].nunique() < 2:
        print(f"WorkerID: {worker_id} has only one class in target variable. Sl
        return {}

    # Standardize the features
    scaler = StandardScaler()
    X = scaler.fit_transform(df[features])
    y = df[target]

    # Train logistic regression model
    model = LogisticRegression()
    model.fit(X, y)

    # Get absolute values of coefficients and normalize them to sum to 1
```

```
    weights = np.abs(model.coef_[0])
    weights /= weights.sum()

    # Return dictionary of feature importance
    return dict(zip(features, weights))
```

In [15]:

```
# Get overall weights
weights = calculate_metric_weights(df)
print(weights)

# Get weights for all writers
unique_workers = df['workerID'].unique()

for worker in unique_workers:
  weights = calculate_metric_weights(df, worker_id=worker)
  print(f'WorkerID: {worker}, Weights: {weights}')
```

{'tone_similarity': np.float64(0.11097236315607205), 'pos_similarity': np.float64(0.39681200787667653), 'coherence_score': np.float64(0.28116097263263495), 'syn_similarity': np.float64(0.02395134361340448), 'ai_coverage': np.float64(0.18710331272121195)}
WorkerID: A2WGW5Y3ZFBDEC, Weights: {'tone_similarity': np.float64(0.2864170480700348), 'pos_similarity': np.float64(0.1499254389361868), 'coherence_score': np.float64(0.00621779346830999), 'syn_similarity': np.float64(0.49844575068780966), 'ai_coverage': np.float64(0.05899396883765867)}
WorkerID: AZCGF2D7QIO10, Weights: {'tone_similarity': np.float64(0.10075451011843206), 'pos_similarity': np.float64(0.2574761033202567), 'coherence_score': np.float64(0.37052214139856876), 'syn_similarity': np.float64(0.2121920890503347), 'ai_coverage': np.float64(0.05905515611240756)}
WorkerID: A345TDMHP3DQ3G, Weights: {'tone_similarity': np.float64(0.11382502647662301), 'pos_similarity': np.float64(0.4139190820820991), 'coherence_score': np.float64(0.2982455136495654), 'syn_similarity': np.float64(0.12834969487294032), 'ai_coverage': np.float64(0.04566068291877205)}
WorkerID: A2W121DQXNQK1, Weights: {'tone_similarity': np.float64(0.3254243122350869), 'pos_similarity': np.float64(0.029414183044931687), 'coherence_score': np.float64(0.3167718925864591), 'syn_similarity': np.float64(0.08150379272430178), 'ai_coverage': np.float64(0.24688581940922064)}
WorkerID: A324VBRLXHG5IB, Weights: {'tone_similarity': np.float64(0.3006255253006611), 'pos_similarity': np.float64(0.2962431350826676), 'coherence_score': np.float64(0.10913433934454694), 'syn_similarity': np.float64(0.09421907593639245), 'ai_coverage': np.float64(0.1997779243357319)}
WorkerID: A2ONILC0LZKG6Y, Weights: {'tone_similarity': np.float64(0.22526671925815853), 'pos_similarity': np.float64(0.07826900266516451), 'coherence_score': np.float64(0.31440099648645475), 'syn_similarity': np.float64(0.24610791662950904), 'ai_coverage': np.float64(0.13595536496071306)}
WorkerID: A3OLRWACCCCUTU, Weights: {'tone_similarity': np.float64(0.09308846675377784), 'pos_similarity': np.float64(0.006299225394154346), 'coherence_score': np.float64(0.4455648081088857), 'syn_similarity': np.float64(0.21545738579782872), 'ai_coverage': np.float64(0.2395901139453533)}
WorkerID: A2EED3HLTA96CP, Weights: {'tone_similarity': np.float64(0.3089681403364952), 'pos_similarity': np.float64(0.11120232182426563), 'coherence_score': np.float64(0.2403186664522991), 'syn_similarity': np.float64(0.23887262903661619), 'ai_coverage': np.float64(0.1006382423503237)}
WorkerID: A2QKAA5YS0P4CI, Weights: {'tone_similarity': np.float64(0.171889299135242), 'pos_similarity': np.float64(0.07265939839611343), 'coherence_score': np.float64(0.30908813700784876), 'syn_similarity': np.float64(0.22898855112139282), 'ai_coverage': np.float64(0.21737461433940283)}
WorkerID: A3S67QA0SQVPUJ, Weights: {'tone_similarity': np.float64(0.5957129432716168), 'pos_similarity': np.float64(0.04204764559909931), 'coherence_score': np.float64(0.14666389660488685), 'syn_similarity': np.float64(0.15209422259036223), 'ai_coverage': np.float64(0.0634812919340348)}
WorkerID: A1198W1SPF1R4, Weights: {'tone_similarity': np.float64(0.4084557851700455), 'pos_similarity': np.float64(0.28318740530269987), 'coherence_score': np.float64(0.21290610737748678), 'syn_similarity': np.float64(0.028560448676903605), 'ai_coverage': np.float64(0.06689025347286415)}
WorkerID: ANCIB6B6EBBIJ, Weights: {'tone_similarity': np.float64(0.0008726640192682955), 'pos_similarity': np.float64(0.159616663772976), 'coherence_score': np.float64(0.35817131176443), 'syn_similarity': np.float64(0.2817796458217043), 'ai_coverage': np.float64(0.19955971462162136)}
WorkerID: AZZA3J049G7R5, Weights: {'tone_similarity': np.float64(0.11293703089314482), 'pos_similarity': np.float64(0.16234376225933533), 'coherence_score': np.float64(0.49714303837148377), 'syn_similarity': np.float64(0.08930613613419593), 'ai_coverage': np.float64(0.13827003234184018)}
WorkerID: A8C3WNWRBWUXO, Weights: {'tone_similarity': np.float64(0.3421139690734312), 'pos_similarity': np.float64(0.03963612665083661), 'coherence_score': np.float64(0.34422766593807164), 'syn_similarity': np.float64(0.08327216424918156), 'ai_coverage': np.float64(0.19075007408847902)}

WorkerID: A1PTH9KTRO06EG, Weights: {'tone_similarity': np.float64(0.1388342231
8461378), 'pos_similarity': np.float64(0.12318162825853199), 'coherence_scor
e': np.float64(0.10551550246026564), 'syn_similarity': np.float64(0.2521152152
9630804), 'ai_coverage': np.float64(0.3803534308002805)}
WorkerID: A2QX3YJXAAHHVV, Weights: {'tone_similarity': np.float64(0.0350107858
3114324), 'pos_similarity': np.float64(0.18915127821222674), 'coherence_scor
e': np.float64(0.29098049928695807), 'syn_similarity': np.float64(0.1242846819
3583758), 'ai_coverage': np.float64(0.36057275473383443)}
WorkerID: A23KAJRDVCVGOE, Weights: {'tone_similarity': np.float64(0.3508241195
0579395), 'pos_similarity': np.float64(0.22089572420603215), 'coherence_scor
e': np.float64(0.02913820655651091), 'syn_similarity': np.float64(0.1058530883
7287803), 'ai_coverage': np.float64(0.2932888613587852)}
WorkerID: A1QUQ0TV9KVD4C, Weights: {'tone_similarity': np.float64(0.1566889269
5480535), 'pos_similarity': np.float64(0.32863594454940215), 'coherence_scor
e': np.float64(0.2996974250023059), 'syn_similarity': np.float64(0.01348367523
770947), 'ai_coverage': np.float64(0.20149402825577714)}
WorkerID: A1VZSFHTU51JP0, Weights: {'tone_similarity': np.float64(0.4039494651
3166957), 'pos_similarity': np.float64(0.34097030228594766), 'coherence_scor
e': np.float64(0.14166153338719917), 'syn_similarity': np.float64(0.0105353200
13226884), 'ai_coverage': np.float64(0.10288337918195666)}
WorkerID: A3MYPYBVHX7FQ2, Weights: {'tone_similarity': np.float64(0.1071720432
8740723), 'pos_similarity': np.float64(0.22448564963844667), 'coherence_scor
e': np.float64(0.12355616105128413), 'syn_similarity': np.float64(0.2219331074
4176004), 'ai_coverage': np.float64(0.3228530385811019)}
WorkerID: ASVRLMDNQBUD9, Weights: {'tone_similarity': np.float64(0.04653852649
6807564), 'pos_similarity': np.float64(0.4963255450188651), 'coherence_score':
np.float64(0.034277348673825696), 'syn_similarity': np.float64(0.2335941938505
239), 'ai_coverage': np.float64(0.1892643859599777)}
WorkerID: A2YTQDLACTLIBA, Weights: {'tone_similarity': np.float64(0.0296758632
80804314), 'pos_similarity': np.float64(0.02506530388696718), 'coherence_scor
e': np.float64(0.29438503450316583), 'syn_similarity': np.float64(0.0659563703
2483724), 'ai_coverage': np.float64(0.5849174280042254)}
WorkerID: A1TW2BZRRS874Z, Weights: {'tone_similarity': np.float64(0.1766257980
4117315), 'pos_similarity': np.float64(0.3497003405012193), 'coherence_score':
np.float64(0.1349599244468325), 'syn_similarity': np.float64(0.1940399740569090
62), 'ai_coverage': np.float64(0.14467396295386561)}
WorkerID: A2OVX9UW5WANQE, Weights: {'tone_similarity': np.float64(0.2560945546
533722), 'pos_similarity': np.float64(0.0018877305841718543), 'coherence_scor
e': np.float64(0.17248088218431035), 'syn_similarity': np.float64(0.0246950482
67015993), 'ai_coverage': np.float64(0.5448417843111296)}
WorkerID: A2OVOVZBJYUO, Weights: {'tone_similarity': np.float64(0.364616784599
4494), 'pos_similarity': np.float64(0.4123783837447544), 'coherence_score': n
p.float64(0.09358432077864522), 'syn_similarity': np.float64(0.047821771038202
02), 'ai_coverage': np.float64(0.08159873983894905)}
WorkerID: A1FVXS8IM5QYO8, Weights: {'tone_similarity': np.float64(0.1784059077
8494414), 'pos_similarity': np.float64(0.2970671196480524), 'coherence_score':
np.float64(0.3343264845314462), 'syn_similarity': np.float64(0.1081744964583600
57), 'ai_coverage': np.float64(0.08202599157719667)}
WorkerID: ABL2FXYMI00T6, Weights: {'tone_similarity': np.float64(0.07487919325
731936), 'pos_similarity': np.float64(0.06338142002686713), 'coherence_score':
np.float64(0.6285275910378962), 'syn_similarity': np.float64(0.0814393859378960
76), 'ai_coverage': np.float64(0.15177240974002035)}
WorkerID: AZLZA0Q87TJZO, Weights: {'tone_similarity': np.float64(0.40041291658
619566), 'pos_similarity': np.float64(0.010002048705241772), 'coherence_scor
e': np.float64(0.3170844018523748), 'syn_similarity': np.float64(0.09393321383
300927), 'ai_coverage': np.float64(0.17856741902317852)}
WorkerID: A3O5RKGH6VB19C, Weights: {'tone_similarity': np.float64(0.1146119385
7007895), 'pos_similarity': np.float64(0.10243566181995697), 'coherence_scor
e': np.float64(0.3610233678105878), 'syn_similarity': np.float64(0.29914326324
333385), 'ai_coverage': np.float64(0.1227857685560425)}

```
WorkerID: A3P9TM5PRYBH90, Weights: {'tone_similarity': np.float64(0.0255028605
4463665), 'pos_similarity': np.float64(0.08394560683440744), 'coherence_scor
e': np.float64(0.4097798240235627), 'syn_similarity': np.float64(0.00477291340
4165938), 'ai_coverage': np.float64(0.4759987951932273)}
WorkerID: A118BQHK3S4UDV, Weights: {'tone_similarity': np.float64(0.3212127251
95445), 'pos_similarity': np.float64(0.0297624819298747), 'coherence_score': n
p.float64(0.39470063818741113), 'syn_similarity': np.float64(0.209032281167704
92), 'ai_coverage': np.float64(0.04529187351956443)}
WorkerID: A17Q4QN6UE0EZC has only one class in target variable. Skipping...
WorkerID: A17Q4QN6UE0EZC, Weights: {}
WorkerID: AFIK3VBMMX6G6 has only one class in target variable. Skipping...
WorkerID: AFIK3VBMMX6G6, Weights: {}
WorkerID: A14OQ52EFQAN2W has only one class in target variable. Skipping...
WorkerID: A14OQ52EFQAN2W, Weights: {}
WorkerID: A1WKF2VH7TV0H2 has only one class in target variable. Skipping...
WorkerID: A1WKF2VH7TV0H2, Weights: {}
WorkerID: A3DUPRZSMU9W5R, Weights: {'tone_similarity': np.float64(0.0481348484
1248044), 'pos_similarity': np.float64(0.3097842612758368), 'coherence_score':
np.float64(0.017304179456697686), 'syn_similarity': np.float64(0.5528286533845
296), 'ai_coverage': np.float64(0.07194805747045538)}
WorkerID: A3DS5B06ZCD3E3, Weights: {'tone_similarity': np.float64(0.1715741158
922311), 'pos_similarity': np.float64(0.007414056394940826), 'coherence_scor
e': np.float64(0.36341440870358166), 'syn_similarity': np.float64(0.3919072800
3457576), 'ai_coverage': np.float64(0.06569013897467059)}
WorkerID: A1E235KE3CSO7H, Weights: {'tone_similarity': np.float64(0.1412963377
3558226), 'pos_similarity': np.float64(0.5753876801476411), 'coherence_score':
np.float64(0.17389492464430592), 'syn_similarity': np.float64(0.00014325879814
947444), 'ai_coverage': np.float64(0.10927779867432108)}
WorkerID: A2RUHO7I7Y4XFA, Weights: {'tone_similarity': np.float64(0.1548090579
1020902), 'pos_similarity': np.float64(0.2323492412138711), 'coherence_score':
np.float64(0.18611836962446915), 'syn_similarity': np.float64(0.19428317428847
125), 'ai_coverage': np.float64(0.23244015696297943)}
WorkerID: A3VEF4M5FIN7KH, Weights: {'tone_similarity': np.float64(0.0799527656
7637451), 'pos_similarity': np.float64(0.35695656244531676), 'coherence_scor
e': np.float64(0.402832031848546), 'syn_similarity': np.float64(0.056568143283
32609), 'ai_coverage': np.float64(0.10369049674643674)}
WorkerID: APRZ7BR8C0ZMQ, Weights: {'tone_similarity': np.float64(0.01659088127
2663578), 'pos_similarity': np.float64(0.272823163721841), 'coherence_score':
np.float64(0.14168436586897346), 'syn_similarity': np.float64(0.39266572069646
42), 'ai_coverage': np.float64(0.17623586844005787)}
WorkerID: A173MXK429XAZQ has only one class in target variable. Skipping...
WorkerID: A173MXK429XAZQ, Weights: {}
WorkerID: A143XRCI1YXAFE, Weights: {'tone_similarity': np.float64(0.1153972039
8601247), 'pos_similarity': np.float64(0.29567809676384055), 'coherence_scor
e': np.float64(0.07808121595865634), 'syn_similarity': np.float64(0.3280587351
238614), 'ai_coverage': np.float64(0.18278474816762927)}
WorkerID: A6KOTWP7N7RLU has only one class in target variable. Skipping...
WorkerID: A6KOTWP7N7RLU, Weights: {}
WorkerID: A23EWFNNOUS10B, Weights: {'tone_similarity': np.float64(0.2867453289
80132), 'pos_similarity': np.float64(0.33055661034811284), 'coherence_score':
np.float64(0.08010976939535645), 'syn_similarity': np.float64(0.07201598274361
272), 'ai_coverage': np.float64(0.23057230853278604)}
WorkerID: A1PBRKFHSF1OF8, Weights: {'tone_similarity': np.float64(0.0627102933
9451119), 'pos_similarity': np.float64(0.43484571596095245), 'coherence_scor
e': np.float64(0.22994530981944442), 'syn_similarity': np.float64(0.2715077728
747926), 'ai_coverage': np.float64(0.0009909079502991953)}
WorkerID: A377LTGWJKY2IW has only one class in target variable. Skipping...
WorkerID: A377LTGWJKY2IW, Weights: {}
WorkerID: A3HE29W5IDR394 has only one class in target variable. Skipping...
WorkerID: A3HE29W5IDR394, Weights: {}
```

```
WorkerID: AM2KK02JXXW48, Weights: {'tone_similarity': np.float64(0.08098369376
397944), 'pos_similarity': np.float64(0.14250209312216586), 'coherence_score':
np.float64(0.32469170933779357), 'syn_similarity': np.float64(0.31327878421822
89), 'ai_coverage': np.float64(0.1385437195578323)}
WorkerID: A17AF42SNQNH9C, Weights: {'tone_similarity': np.float64(0.2417515615
5020944), 'pos_similarity': np.float64(0.30497016127870274), 'coherence_scor
e': np.float64(0.08900057903431974), 'syn_similarity': np.float64(0.0309762071
80510425), 'ai_coverage': np.float64(0.3333014909562576)}
WorkerID: A394JO4NEPCY3M has only one class in target variable. Skipping...
WorkerID: A394JO4NEPCY3M, Weights: {}
```

In [16]:
```python
def calculate_weights_for_all_workers(df, unique_workers):
  '''
  Calculate the weight of each helpfulness metric for all writers.

  Input: csv file

  Output: dictionary of dictionary of feature importance
  '''

  worker_weights = {}
  for worker in unique_workers:
      weights = calculate_metric_weights(df, worker_id=worker)
      worker_weights[worker] = weights

  return worker_weights
```

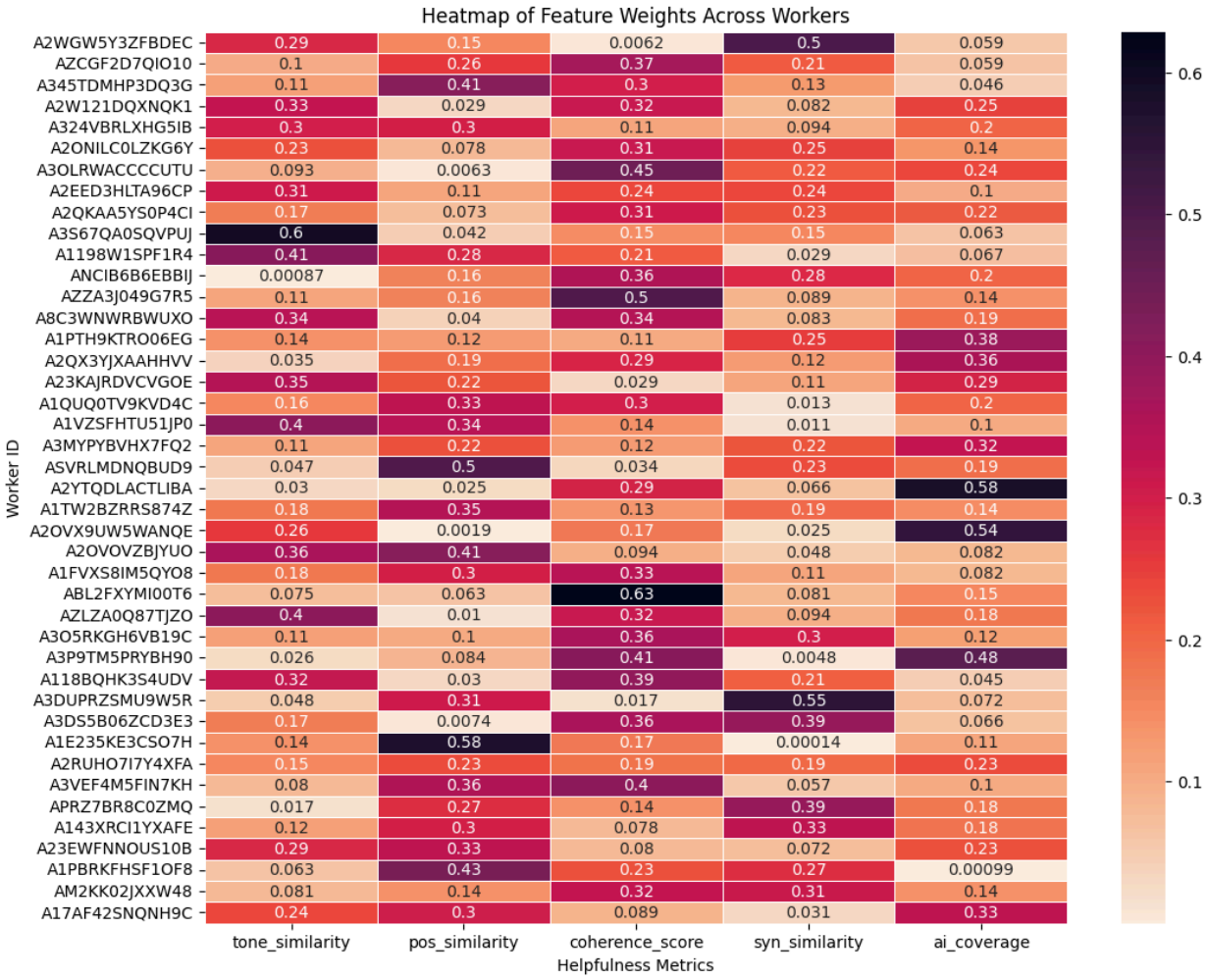
In [17]:
```python
import seaborn as sns
import matplotlib.pyplot as plt

worker_weights = calculate_weights_for_all_workers(df, unique_workers)
df_weights = pd.DataFrame.from_dict(worker_weights, orient='index')

plt.figure(figsize=(12, 10))
sns.heatmap(df_weights, annot=True, cmap="rocket_r", linewidths=0.5)
plt.xlabel("Helpfulness Metrics")
plt.ylabel("Worker ID")
plt.title("Heatmap of Feature Weights Across Workers")

plt.savefig('heatmap.png', dpi=300)
plt.show()
```

```
WorkerID: A17Q4QN6UE0EZC has only one class in target variable. Skipping...
WorkerID: AFIK3VBMMX6G6 has only one class in target variable. Skipping...
WorkerID: A14OQ52EFQAN2W has only one class in target variable. Skipping...
WorkerID: A1WKF2VH7TV0H2 has only one class in target variable. Skipping...
WorkerID: A173MXK429XAZQ has only one class in target variable. Skipping...
WorkerID: A6KOTWP7N7RLU has only one class in target variable. Skipping...
WorkerID: A377LTGWJKY2IW has only one class in target variable. Skipping...
WorkerID: A3HE29W5IDR394 has only one class in target variable. Skipping...
WorkerID: A394JO4NEPCY3M has only one class in target variable. Skipping...
```

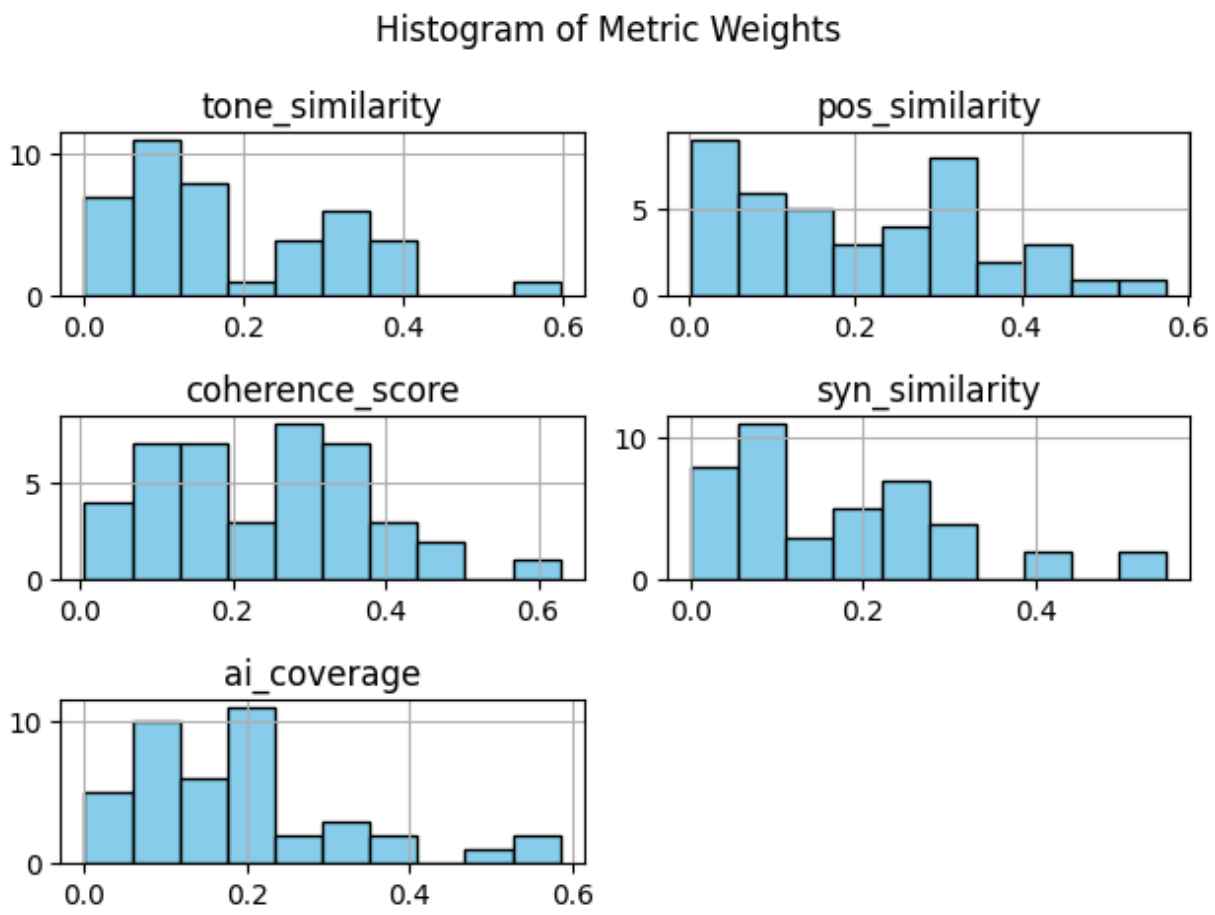


In [18]:
```
# histogram of metric weights
plt.figure(figsize=(4, 10))
df_weights.hist(bins=10, color='skyblue', edgecolor='black')
plt.suptitle('Histogram of Metric Weights', fontsize=12)
plt.xlabel('Weight', fontsize=10)
plt.ylabel('Frequency', fontsize=10)

#save img
plt.savefig('metric_weights_histogram.png')

plt.tight_layout()
plt.show()
```

```
<Figure size 400x1000 with 0 Axes>
```



In [19]:
```
# finding mean and std for each weight
df_weights.describe()
```

Out[19]:

	tone_similarity	pos_similarity	coherence_score	syn_similarity	ai_coverage
count	42.000000	42.000000	42.000000	42.000000	42.000000
mean	0.189699	0.206304	0.243384	0.173195	0.187418
std	0.136424	0.150354	0.143830	0.134338	0.133411
min	0.000873	0.001888	0.006218	0.000143	0.000991
25%	0.084010	0.074062	0.126407	0.074372	0.086679
50%	0.155749	0.205024	0.265650	0.140222	0.164004
75%	0.297155	0.308581	0.341752	0.244299	0.231973
max	0.595713	0.575388	0.628528	0.552829	0.584917

In [12]:
```
from sklearn.cluster import KMeans
# Apply KMeans clustering
kmeans = KMeans(n_clusters=6, random_state=42)
df_weights['Cluster'] = kmeans.fit_predict(df_weights)

plt.figure(figsize=(10, 6))
sns.scatterplot(x=df_weights.iloc[:, 0], y=df_weights.iloc[:, 1], hue=df_weigh
plt.xlabel("First PC")
plt.ylabel("Second PC")
plt.title("Clusters of Workers Based on Helpfulness Metrics")
```

```
plt.legend(title="Cluster")
plt.show()
```