Engram Scoring Empirical Basis

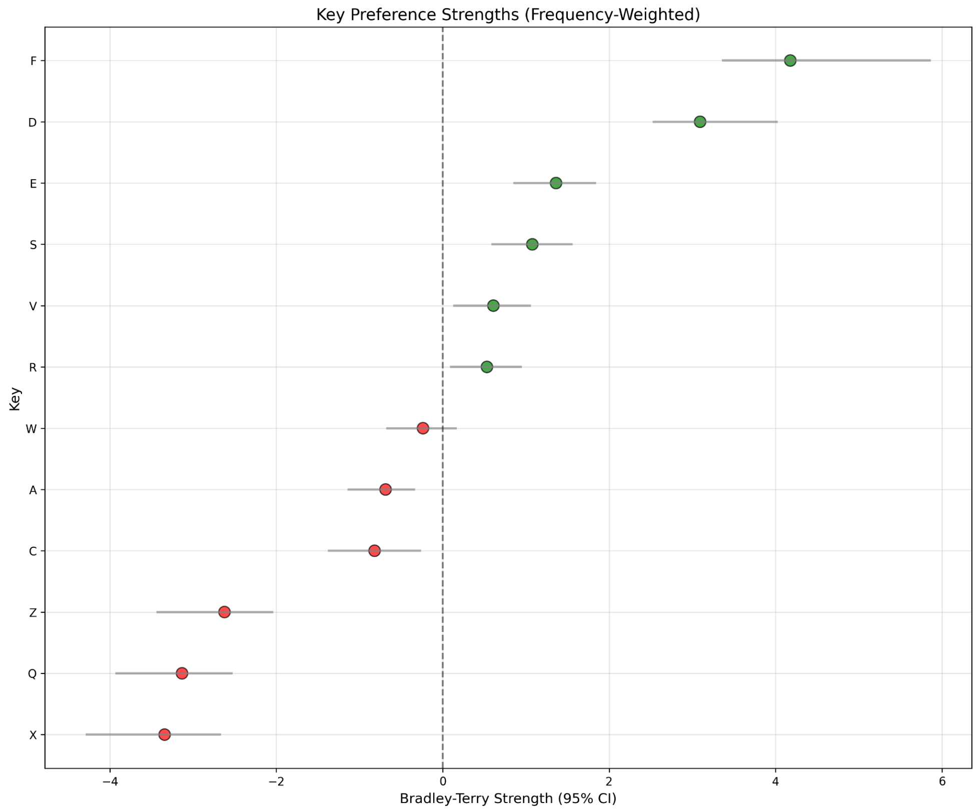
(1) Same letter preferences

Key preferences from same-letter bigram comparisons only (pure key quality)

NON-PROLIFIC DATA

Instances analyzed: 756
Key,Finger,Row,Column,Rank,BT_Strength,CI_Lower,CI_Upper
F,4,2,4,1,4.176477899294208,3.3694946159462913,5.8518218910794975
D,3,2,3,2,3.0925605015169326,2.535203398561629,4.011195656581959
E,3,1,3,3,1.3610655649049088,0.8627562755537985,1.8292961310745344
S,2,2,2,4,1.075787479685951,0.5965120189690969,1.5451887560456279
V,4,3,4,5,0.6082927624618688,0.13806885990143172,1.0451377438105673
R,4,1,4,6,0.5305846239524998,0.09785581151606763,0.9379340920126836
W,2,1,2,7,-0.23776116274762676,-0.6665497625072652,0.15511912266908917
A,1,2,1,8,-0.6876316179656282,-1.1288479091387889,-0.34549849017379886
C,3,3,3,9,-0.81964409295591,-1.3666173205652683,-0.2738513771452572
Z,1,3,1,10,-2.6231357621806337,-3.428717330109768,-2.052629567673498
Q,1,1,1,11,-3.1336203993642586,-3.9212349220570277,-2.5407626980491687
X,2,3,2,12,-3.342975796602313,-4.2759012170275215,-2.679634504460716

[F], [D], [E,S], [V,R], [W], [A,C], [Z,Q,X]

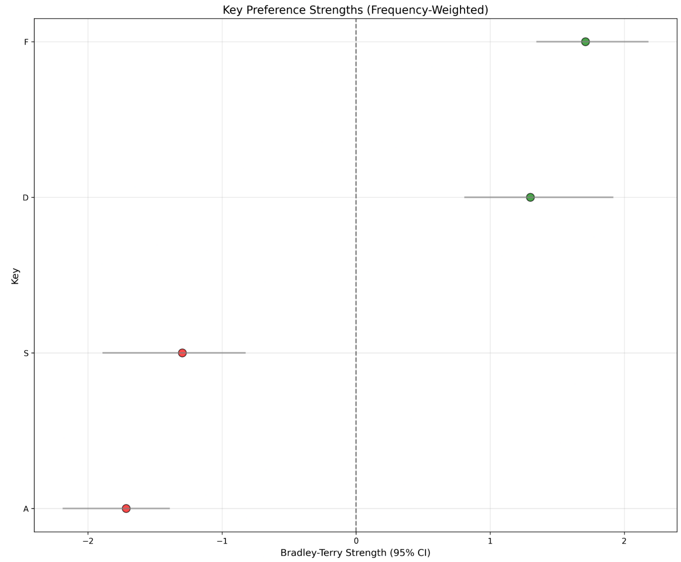


PROLIFIC DATA

Instances analyzed: 426

Key,Finger,Row,Column,Rank,BT_Strength,CI_Lower,CI_Upper
F,4,2,4,1,1.7108928231142055,1.349997568781515,2.1745994361433114
D,3,2,3,2,1.2999441435448653,0.8123428252045043,1.9132785220474997
S,2,2,2,3,-1.295943804524782,-1.8862886514585542,-0.829382224801393
A,1,2,1,4,-1.7148931621342884,-2.183682185233311,-1.3952477892132804

[F,D], [S,A]



(2) Pairwise preferences

28 pairwise key comparisons for detailed analysis

These results rely on mixed-letter bigram data, and are therefore less reliable than the same-letter bigram preference data above.

NON-PROLIFIC DATA

A > Q:
Preference rate: 98.2% (effect size: 48.2%)
95% CI: [93.7%, 99.5%]
Statistical test: p = 0.0000 *** (n=112)

…

```
Key1,Key2,Key1_Finger,Key1_Row,Key1_Column,Key2_Finger,Key2_Row,Key2_Column,Key1_Preference_Rate,CI_Lower,CI_Upper,P_Value,N_Instances,
Effect_Size,Favored_Key,Strength_of_Preference
A,Q,1,2,1,1,1,1,0.9821428571428571,0.9372195125788385,0.9950891884637512,0.0,112,0.4821428571428571,A,0.9821428571428571
F,V,4,2,4,4,3,4,0.9117647058823529,0.8205623087685489,0.9589318846855107,1.113598102620017e-11,68,0.4117647058823529,F,0.9117647058823529
Q,Z,1,1,1,1,3,1,0.1111111111111111,0.06734979672470168,0.17788357565034152,0.0,126,0.3888888888888889,Z,0.8888888888888888
D,V,3,2,3,4,3,4,0.873015873015873,0.8036562871490389,0.9203035393483685,0.0,126,0.373015873015873,D,0.873015873015873
Q,C,1,1,1,3,3,3,0.1951219512195122,0.1417413411718478,0.2624583082912178,5.773159728050814e-15,164,0.3048780487804878,C,0.8048780487804879
A,Z,1,2,1,1,3,1,0.782608695652174,0.7066450273829521,0.8432647144408545,3.1410651857299854e-11,138,0.28260869565217395,A,0.782608695652174
W,C,2,1,2,3,3,3,0.2535211267605634,0.18911246066493514,0.33091428230626607,4.2471193228976745e-09,142,0.24647887323943662,C,0.7464788732394366
S,W,2,2,2,2,1,2,0.7121212121212122,0.629746562998158,0.7824987153297052,1.0926445168646381e-06,132,0.21212121212121215,S,0.7121212121212122
Q,X,1,1,1,2,3,2,0.29411764705882354,0.22404267545649473,0.3755038361985098,1.5711978242904934e-06,136,0.20588235294117646,X,0.7058823529411764
Z,W,1,3,1,2,1,2,0.3380281690140845,0.26540388399745385,0.41918511852647833,0.00011327837928054585,142,0.1619718309859155,W,0.6619718309859155
D,E,3,2,3,3,1,3,0.6567164179104478,0.598015960338678,0.7109876786368109,2.8802831386620653e-07,268,0.15671641791044777,D,0.6567164179104478
```

```
W,X,2,1,2,2,3,2,0.6557377049180327,0.5678346242942165,0.7341326316174024,0.0005809444435782574,122,0.15573770491803274,W,0.6557377049180327
C,X,3,3,3,2,3,2,0.632183908045977,0.5584001643279263,0.7002571830928391,0.00048801311582136186,174,0.13218390804597702,C,0.632183908045977
R,E,4,1,4,3,1,3,0.36923076923076925,0.2911386855184367,0.4548294151016454З,0.002863745566120679,130,0.13076923076923075,E,0.6307692307692307
R,V,4,1,4,4,3,4,0.375,0.3001424864299469,0.45635342285581226,0.002699796063260207,144,0.125,V,0.625
D,R,3,2,3,4,1,4,0.6170212765957447,0.5458498792456085,0.6835061748132011,0.0013318368153214521,188,0.11702127659574468,D,0.6170212765957447
D,S,3,2,3,2,2,2,0.6144578313253012,0.5386237528178108,0.6851143159897441,0.0031842044619818655,166,0.11445783132530118,D,0.6144578313253012
A,W,1,2,1,2,1,2,0.6,0.522604836968219,0.67270592399748,0.011412036386001745,160,0.09999999999999998,A,0.6
S,V,2,2,2,4,3,4,0.5783132530120482,0.5022526711184665,0.650831270697229,0.04359216045052672,166,0.07831325301204817,S,0.5783132530120482
F,D,4,2,4,3,2,3,0.43157894736842106,0.3827096886440501,0.4818177124351257,0.007640794824709918,380,0.06842105263157894,D,0.5684210526315789
E,V,3,1,3,4,3,4,0.5576923076923077,0.4897570420445255,0.6235352291360369,0.09609232945567325,208,0.05769230769230771,E,0.5576923076923077
S,A,2,2,2,1,2,1,0.4489795918367347,0.354283622838111,0.5475245394192032,0.31242221124269354,98,0.05102040816326531,A,0.5510204081632653
S,R,2,2,2,4,1,4,0.4507042253521127,0.37123963199849236,0.532765716595748,0.2400531710910312,142,0.049295774647887314,R,0.5492957746478873
W,Q,2,1,2,1,1,1,0.5294117647058824,0.4124092208872794,0.6432689357880106,0.6276258050283592,68,0.02941176470588236,W,0.5294117647058824
F,R,4,2,4,4,1,4,0.4857142857142857,0.4044181106648916,0.5677734950599487,0.7353166906373405,140,0.01428571428571429,R,0.5142857142857142
X,Z,2,3,2,1,3,1,0.5096153846153846,0.44212262773952915,0.5767594174572312,0.781511294998714,208,0.009615384615384581,X,0.5096153846153846
S,E,2,2,2,3,1,3,0.5070422535211268,0.4257166400412402,0.587996881588154,0.8667120865723021,142,0.007042253521126751,S,0.5070422535211268
F,E,4,2,4,3,1,3,0.4964788732394366,0.45552275121932617,0.5374823029992868,0.8667120865723021,568,0.0035211267605633756,E,0.5035211267605634
```

PROLIFIC DATA

A > Z:

Preference rate: 87.7% (effect size: 37.7%)

95% CI: [83.1%, 91.1%]

Statistical test: p = 0.0000 *** (n=260)

…

```
Key1,Key2,Key1_Finger,Key1_Row,Key1_Column,Key2_Finger,Key2_Row,Key2_Column,Key1_Preference_Rate,CI_Lower,CI_Upper,P_Value,N_Instances,Effect_Size,Favored_Key,Strength_of_Preference
A,Z,1,2,1,1,3,1,0.8769230769230769,0.8314160184452871,0.9114543434609551,0.0,260,0.3769230769230769,A,0.8769230769230769
F,V,4,2,4,4,3,4,0.8658008658008658,0.831684012805019,0.8938847265160098,0.0,462,0.3658008658008658,F,0.8658008658008658
E,V,3,1,3,4,3,4,0.8541666666666666,0.7699818497355329,0.9110979421986138,3.914868429433227e-12,96,0.35416666666666663,E,0.8541666666666666
W,X,2,1,2,2,3,2,0.8360655737704918,0.7844722420623357,0.8772410999279933,0.0,244,0.33606557377049184,W,0.8360655737704918
A,Q,1,2,1,1,1,1,0.8186813186813187,0.7758400305506358,0.8548664720382672,0.0,364,0.31868131868131866,A,0.8186813186813187
Z,W,1,3,1,2,1,2,0.1917808219178082,0.13615172907824855,0.263213435220523,9.43689570931383e-14,146,0.3082191780821918,W,0.8082191780821918
C,X,3,3,3,2,3,2,0.7626262626262627,0.7183229606407648,0.8018832246144667,0.0,396,0.26262626262626265,C,0.7626262626262627
Q,X,1,1,1,2,3,2,0.7169811320754716,0.6248100335076636,0.7939753731491312,7.898568900044367e-06,106,0.21698113207547165,Q,0.7169811320754716
D,E,3,2,3,3,1,3,0.2991913746630728,0.2673469832477767,0.33310429701332567,0.0,742,0.20080862533692723,E,0.7008086253369272
S,R,2,2,2,4,1,4,0.7,0.5456998118185507,0.8192515477025347,0.011412036386001745,40,0.19999999999999996,S,0.7
W,Q,2,1,2,1,1,1,0.6987951807228916,0.6474019396789825,0.7456406598366265,4.3431924723336124e-13,332,0.1987951807228916,W,0.6987951807228916
Q,C,1,1,1,3,3,3,0.33544303797468356,0.2856381702065602,0.38920073159352414,4.902239503223882e-09,316,0.16455696202531644,C,0.6645569620253164
F,E,4,2,4,3,1,3,0.6643356643356644,0.6077496502963156,0.7165655818394421,2.7235788513380044e-08,286,0.16433566433566438,F,0.6643356643356644
```

```
S,E,2,2,2,3,1,3,0.3409090909090909,0.2503913492660681,0.44473544467276116,0.002837544888792287,88,0.15909090909090912,E,0.6590909090909092
X,Z,2,3,2,1,3,1,0.6381578947368421,0.5827267238132403,0.6901410059649207,1.452003717483663e-06,304,0.13815789473684215,X,0.6381578947368421
W,C,2,1,2,3,3,3,0.36231884057971014,0.3078601475225368,0.4205575061007731,4.769884588862183e-06,276,0.13768115942028986,C,0.6376811594202898
F,R,4,2,4,4,1,4,0.6358490566037736,0.6064390763602426,0.6642779535243212,0.0,1060,0.13584905660377355,F,0.6358490566037736
S,W,2,2,2,2,1,2,0.632016632016632,0.6010730293995815,0.6619100903249927,2.220446049250313e-16,962,0.1320166320166320 5,S,0.632016632016632
R,E,4,1,4,3,1,3,0.38387096774193546,0.34641663111109544,0.4227554911261525,7.33019178689176e-09,620,0.11612903225806454,E,0.6161290322580646
F,D,4,2,4,3,2,3,0.615686274509804,0.5727537823549704,0.656889034460199,1.740348403167502e-07,510,0.11568627450980395,F,0.615686274509804
D,V,3,2,3,4,3,4,0.6078431372549019,0.5394445112263195,0.6722553115600429,0.0020657274310063034,204,0.10784313725490191,D,0.6078431372549019
S,V,2,2,2,4,3,4,0.5797101449275363,0.49629083327975365,0.6588119144141406,0.06110093043955156,138,0.07971014492753625,S,0.5797101449275363
D,S,3,2,3,2,2,2,0.45,0.40776251841504935,0.4929708062876847,0.022586888164179753,520,0.04999999999999999,S,0.55
A,W,1,2,1,2,1,2,0.4672897196261682,0.4016093613033394,0.534123716891278,0.33855701001809946,214,0.03271028037383178,W,0.5327102803738317
S,A,2,2,2,1,2,1,0.4702702702702703,0.41998010839439337,0.5211714159191476,0.2527374785444083,370,0.029729729729729704,A,0.5297297297297296
D,R,3,2,3,4,1,4,0.5274261603375527,0.48244185861597116,0.5719694937689133,0.23239209697335994,474,0.02742616033755274,D,0.5274261603375527
Q,Z,1,1,1,1,3,1,0.5217391304347826,0.4672441868636643,0.5757214935953836,0.43527934321814277,322,0.021739130434782594,Q,0.5217391304347826
R,V,4,1,4,4,3,4,0.4966887417218543,0.440695185444668,0.5527654787579163,0.908376110406135,302,0.0033112582781457123,V,0.5033112582781457
```

Focus on non-Prolific data

Prolific data only has same-letter bigram typing data for four of the twelve left-hand finger-column keys.

Non-Prolific data suggests S is a preferred key while Prolific data suggests it's disliked. This magnitude of difference points to Prolific's potential sample bias or data quality issues.

- Non-Prolific data: S = +1.08 (rank 4, positive preference)
- Prolific data: S = -1.30 (rank 3, but negative preference)

For these reasons, we will focus on non-Prolific data for key preferences.

Statistical separability

Statistical overlap (overlapping CIs):

E & S: CIs overlap substantially, should be grouped
V & R: CIs overlap heavily, should be grouped
A & C: CIs overlap, should be grouped
Z, Q & X: All overlap, bottom cluster

Borderline cases:

W sits alone, CI spans zero (neutral)

Natural break points in non-Prolific BT strength distribution (non-overlapping CIs):

F: 4.18 ← Isolated top performer
D: 3.09 ← Clear second tier
E: 1.36, S: 1.08 ← Positive cluster

V: 0.61, R: 0.53 ← Near-neutral cluster
W: -0.238 ← Isolated slightly negative
A: -0.688, C: -0.820 ← Negative cluster
Z: -2.62, Q: -3.13, X: -3.34 ← Very negative cluster

Cluster averages

- Tier 1 (F): 4.176 BT strength - Elite performance
- Tier 2 (D): 3.093 BT strength - Strong performance
- Tier 3 (E,S): 1.218 BT strength - Positive performance
- Tier 4 (V,R): 0.569 BT strength - Neutral performance
- Tier 5 (W): -0.238 BT strength - Negative performance (isolated)
- Tier 6 (A,C): -0.754 BT strength - Negative performance
- Tier 7 (Z,Q,X): -3.033 BT strength - Very poor performance

7-tier scaling

```python

# Bradley-Terry Strength to MOO Values Transformation
import numpy as np

# Step 1: Individual key Bradley-Terry strengths
individual_bt_values = {
    'F': 4.176477899294208,
    'D': 3.0925605015169326,
    'E': 1.3610655649049088,
    'S': 1.075787479685951,
    'V': 0.6082927624618688,
    'R': 0.5305846239524998,
    'W': -0.23776116274762676,
    'A': -0.6876316179656282,
    'C': -0.81964409295591,
    'Z': -2.6231357621806337,
    'Q': -3.1336203993642586,
    'X': -3.342975796602313
}

# Step 2: Find min/max range from individual keys
min_bt = min(individual_bt_values.values())  # X: -3.343
max_bt = max(individual_bt_values.values())  # F: 4.176
range_bt = max_bt - min_bt

print("STEP 1: Establish BT Range")
print(f"Min BT: {min_bt:.3f} (key X)")
print(f"Max BT: {max_bt:.3f} (key F)")
print(f"Range: {range_bt:.3f}")
print()
```

```
# Step 3: Define cluster averages based on statistical analysis
tier_bt_values = {
    'Tier 1 (F)': 4.176,
    'Tier 2 (D)': 3.093,
    'Tier 3 (E,S)': 1.218,
    'Tier 4 (V,R)': 0.569,
    'Tier 5 (W)': -0.238,
    'Tier 6 (A,C)': -0.754,
    'Tier 7 (Z,Q,X)': -3.033
}

# Step 4: Normalization functions
def linear_normalize(bt_value, min_bt, range_bt):
    """Linear normalization to 0-1 scale"""
    return (bt_value - min_bt) / range_bt

def conservative_normalize(bt_value, min_bt, range_bt):
    """Conservative scaling to 0.1-1.0 range to avoid zero values"""
    linear = linear_normalize(bt_value, min_bt, range_bt)
    return 0.1 + 0.9 * linear

# Step 5: Apply transformations
print("STEP 2: Transform Cluster Averages")
print("Formula: Linear = (cluster_avg - min_individual) / range_individual")
print("Formula: Conservative = 0.1 + 0.9 * linear")
print()

moo_values_linear = {}
moo_values_conservative = {}

for tier, bt_value in tier_bt_values.items():
    linear = linear_normalize(bt_value, min_bt, range_bt)
    conservative = conservative_normalize(bt_value, min_bt, range_bt)

    moo_values_linear[tier] = linear
    moo_values_conservative[tier] = conservative

    print(f"{tier}: {bt_value:.3f}")
    print(f"  Linear: ({bt_value:.3f} - {min_bt:.3f}) / {range_bt:.3f} = {linear:.3f}")
    print(f"  Conservative: 0.1 + 0.9 × {linear:.3f} = {conservative:.3f}")
    print()

# Step 6: Calculate 7-tier cluster averages from individual key BT values
print("STEP 3: Calculate 7-Tier Cluster Averages")

# Using exact individual BT values for cluster averages
cluster_7_calculations = {
    'Tier 1 (F)': [individual_bt_values['F']],  # F alone
    'Tier 2 (D)': [individual_bt_values['D']],  # D alone
```

```
    'Tier 3 (E,S)': [individual_bt_values['E'], individual_bt_values['S']],  # E, S
    'Tier 4 (V,R)': [individual_bt_values['V'], individual_bt_values['R']],  # V, R
    'Tier 5 (W)': [individual_bt_values['W']],  # W alone (isolated)
    'Tier 6 (A,C)': [individual_bt_values['A'], individual_bt_values['C']],  # A, C
    'Tier 7 (Z,Q,X)': [individual_bt_values['Z'], individual_bt_values['Q'], individual_bt_values['X']]  # Z, Q, X
}

# Calculate actual cluster averages
tier_7_bt_values = {}
for tier, bt_list in cluster_7_calculations.items():
    avg = sum(bt_list) / len(bt_list)
    tier_7_bt_values[tier] = avg
    keys_in_tier = [k for k, v in individual_bt_values.items() if v in bt_list]
    print(f"{tier}: {keys_in_tier} → Average BT = {avg:.3f}")
print()

# Step 7: Apply conservative normalization to 7-tier averages
print("STEP 4: Apply Conservative Normalization to 7-Tier Averages")
tier_7_moo_values = {}

for tier, bt_value in tier_7_bt_values.items():
    linear = linear_normalize(bt_value, min_bt, range_bt)
    conservative = conservative_normalize(bt_value, min_bt, range_bt)
    tier_7_moo_values[tier] = conservative

    print(f"{tier}: {bt_value:.3f}")
    print(f"  Linear: ({bt_value:.3f} - {min_bt:.3f}) / {range_bt:.3f} = {linear:.3f}")
    print(f"  Conservative: 0.1 + 0.9 × {linear:.3f} = {conservative:.3f}")
    print()

# Step 8: Generate final 7-tier MOO values
print("STEP 5: Final 7-Tier MOO Values")
print("tier_values = {")

key_to_tier_7 = {
    'F': 'Tier 1 (F)',
    'D': 'Tier 2 (D)',
    'E': 'Tier 3 (E,S)',
    'S': 'Tier 3 (E,S)',
    'V': 'Tier 4 (V,R)',
    'R': 'Tier 4 (V,R)',
    'W': 'Tier 5 (W)',
    'A': 'Tier 6 (A,C)',
    'C': 'Tier 6 (A,C)',
    'Z': 'Tier 7 (Z,Q,X)',
    'Q': 'Tier 7 (Z,Q,X)',
    'X': 'Tier 7 (Z,Q,X)'
}
```

```
for key in ['F', 'D', 'E', 'S', 'V', 'R', 'W', 'A', 'C', 'Z', 'Q', 'X']:
    tier = key_to_tier_7[key]
    value = tier_7_moo_values[tier]
    print(f"    '{key}': {value:.3f},")
print("}")

print("\n7-Tier Insights:")
print("• W is isolated from A/C based on non-overlapping confidence intervals")
print("• Better reflects statistical separability from your Bradley-Terry analysis")
print("• Each tier represents statistically distinct performance levels")
print("• Conservative scaling ensures all values remain active in MOO optimization")
```

STEP 1: Establish BT Range
Min BT: -3.343 (key X)
Max BT: 4.176 (key F)
Range: 7.519

STEP 2: Transform Cluster Averages
Formula: Linear = (cluster_avg - min_individual) / range_individual
Formula: Conservative = 0.1 + 0.9 * linear

Tier 1 (F): 4.176
  Linear: (4.176 - -3.343) / 7.519 = 1.000
  Conservative: 0.1 + 0.9 × 1.000 = 1.000

Tier 2 (D): 3.093
  Linear: (3.093 - -3.343) / 7.519 = 0.856
  Conservative: 0.1 + 0.9 × 0.856 = 0.870

Tier 3 (E,S): 1.218
  Linear: (1.218 - -3.343) / 7.519 = 0.607
  Conservative: 0.1 + 0.9 × 0.607 = 0.646

Tier 4 (V,R): 0.569
  Linear: (0.569 - -3.343) / 7.519 = 0.520
  Conservative: 0.1 + 0.9 × 0.520 = 0.568

Tier 5 (W): -0.238
  Linear: (-0.238 - -3.343) / 7.519 = 0.413
  Conservative: 0.1 + 0.9 × 0.413 = 0.472

Tier 6 (A,C): -0.754
  Linear: (-0.754 - -3.343) / 7.519 = 0.344
  Conservative: 0.1 + 0.9 × 0.344 = 0.410

Tier 7 (Z,Q,X): -3.033
  Linear: (-3.033 - -3.343) / 7.519 = 0.041
  Conservative: 0.1 + 0.9 × 0.041 = 0.137

STEP 3: Calculate 7-Tier Cluster Averages
Tier 1 (F): ['F'] → Average BT = 4.176
Tier 2 (D): ['D'] → Average BT = 3.093
Tier 3 (E,S): ['E', 'S'] → Average BT = 1.218
Tier 4 (V,R): ['V', 'R'] → Average BT = 0.569
Tier 5 (W): ['W'] → Average BT = -0.238
Tier 6 (A,C): ['A', 'C'] → Average BT = -0.754
Tier 7 (Z,Q,X): ['Z', 'Q', 'X'] → Average BT = -3.033

STEP 4: Apply Conservative Normalization to 7-Tier Averages
Tier 1 (F): 4.176
  Linear: (4.176 - -3.343) / 7.519 = 1.000
  Conservative: 0.1 + 0.9 × 1.000 = 1.000

Tier 2 (D): 3.093
  Linear: (3.093 - -3.343) / 7.519 = 0.856
  Conservative: 0.1 + 0.9 × 0.856 = 0.870

Tier 3 (E,S): 1.218
  Linear: (1.218 - -3.343) / 7.519 = 0.607
  Conservative: 0.1 + 0.9 × 0.607 = 0.646

Tier 4 (V,R): 0.569
  Linear: (0.569 - -3.343) / 7.519 = 0.520
  Conservative: 0.1 + 0.9 × 0.520 = 0.568

Tier 5 (W): -0.238
  Linear: (-0.238 - -3.343) / 7.519 = 0.413
  Conservative: 0.1 + 0.9 × 0.413 = 0.472

Tier 6 (A,C): -0.754
  Linear: (-0.754 - -3.343) / 7.519 = 0.344
  Conservative: 0.1 + 0.9 × 0.344 = 0.410

Tier 7 (Z,Q,X): -3.033
  Linear: (-3.033 - -3.343) / 7.519 = 0.041
  Conservative: 0.1 + 0.9 × 0.041 = 0.137

STEP 5: Final 7-Tier MOO Values
```
tier_values = {
    'F': 1.000,
    'D': 0.870,
    'E': 0.646,
    'S': 0.646,
    'V': 0.568,
    'R': 0.568,
    'W': 0.472,
    'A': 0.410,
```

'C': 0.410,
'Z': 0.137,
'Q': 0.137,
'X': 0.137,
}

(3) ROW SEPARATION

Preferences for smaller row separation distances

NON-PROLIFIC DATA

Instances analyzed: 1156

OVERALL ROW SEPARATION PREFERENCE:
Preference rate: 77.5% favor smaller distances (effect size: 27.5%)
95% CI: [75.0%, 79.8%]
Statistical test: p = 0.0000 *** (n=1156)

Same Row (0) vs Reach (1):
Preference: 65.7% favor Smaller distance (effect size: 15.7%)
95% CI: [61.5%, 69.8%]
Statistical test: p = 0.0000 *** (n=502)

Reach (1) vs Hurdle (2):
Preference: 86.5% favor Smaller distance (effect size: 36.5%)
95% CI: [83.7%, 88.9%]
Statistical test: p = 0.0000 *** (n=654)

PROLIFIC DATA

OVERALL ROW SEPARATION PREFERENCE:
Preference rate: 73.5% favor smaller distances (effect size: 23.5%)
95% CI: [71.9%, 75.0%]
Statistical test: p = 0.0000 *** (n=3038)

Same Row (0) vs Reach (1):
Preference: 71.1% favor Smaller distance (effect size: 21.1%)
95% CI: [68.8%, 73.3%]
Statistical test: p = 0.0000 *** (n=1576)

Reach (1) vs Hurdle (2):
Preference: 76.1% favor Smaller distance (effect size: 26.1%)
95% CI: [73.8%, 78.2%]
Statistical test: p = 0.0000 *** (n=1462)

Sample-size weighted meta-analysis

Consistent direction: Both datasets show the same preference ordering (same row > reach > hurdle).
Different magnitudes: Prolific shows more moderate effect sizes.

```python
def combine_datasets(dataset1, dataset2):
    # Combine effect sizes weighted by sample size and inverse variance

    # Non-Prolific data
    n1_same_reach = 502
    effect1_same_reach = 0.157
    ci1_same_reach_width = 0.698 - 0.615  # 0.083

    n1_reach_hurdle = 654
    effect1_reach_hurdle = 0.365
    ci1_reach_hurdle_width = 0.889 - 0.837  # 0.052

    # Convert to cumulative cost structure
    cost_same_row = 0.0
    cost_reach = 0.157      # effect size
    cost_hurdle = 0.157 + 0.365  # 0.522  # Cumulative

    # Convert to scores (flip and normalize to 0-1 range)
    max_cost = 0.522
    score_same_row = 1.0
    score_reach = 1.0 - (0.157 / 0.522) = 0.699
    score_hurdle = 1.0 - (0.522 / 0.522) = 0.0

    # Prolific data
    n2_same_reach = 1576
    effect2_same_reach = 0.211
    ci2_same_reach_width = 0.733 - 0.688  # 0.045

    n2_reach_hurdle = 1462
    effect2_reach_hurdle = 0.261
    ci2_reach_hurdle_width = 0.782 - 0.738  # 0.044

    # Weight by sample size and confidence (inverse of CI width)
    def weighted_average(effect1, n1, ci_width1, effect2, n2, ci_width2):
        weight1 = n1 / ci_width1
        weight2 = n2 / ci_width2
        return (effect1 * weight1 + effect2 * weight2) / (weight1 + weight2)

    combined_same_reach = weighted_average(
        effect1_same_reach, n1_same_reach, ci1_same_reach_width,
        effect2_same_reach, n2_same_reach, ci2_same_reach_width
    )
```

```
    combined_reach_hurdle = weighted_average(
        effect1_reach_hurdle, n1_reach_hurdle, ci1_reach_hurdle_width,
        effect2_reach_hurdle, n2_reach_hurdle, ci2_reach_hurdle_width
    )

    return combined_same_reach, combined_reach_hurdle

# Results: 0.203 for same→reach, 0.290 for reach→hurdle

# Convert to cumulative cost structure
cost_same_row = 0.0
cost_reach = 0.203      # Combined effect size
cost_hurdle = 0.203 + 0.290 = 0.493  # Cumulative

# Convert to scores (flip and normalize to 0-1 range)
max_cost = 0.493
score_same_row = 1.0
score_reach = 1.0 - (0.203 / 0.493) = 0.588
score_hurdle = 1.0 - (0.493 / 0.493) = 0.0
```

(4) COLUMN SEPARATION

Column separation analysis with row controls + reach vs hurdle by column pattern

METHODS NOTES:
- Same-vs-other column tests exclude same-row bigrams (row separation = 0)
- All comparisons control for row separation (1-row vs 1-row, 2-row vs 2-row)
- Adjacent-vs-distant tests separated by row pattern for precision

NON-PROLIFIC DATA

Instances analyzed: 750

OVERALL COLUMN SEPARATION PREFERENCE (WITH ROW CONTROLS):
Preference rate: 64.5% favor smaller distances (effect size: 14.5%)
95% CI: [61.0%, 67.9%]
Statistical test: p = 0.0000 *** (n=750)

SAME COLUMN (0) VS ADJACENT COLUMN (1) - ROW CONTROLLED:
(Excludes same-row bigrams per methodology)

Reach Movements (1 row apart):
Preference: 59.8% favor same column (effect size: 9.8%)
95% CI: [52.3%, 66.8%]
Statistical test: p = 0.0100 ** (n=174)

Hurdle Movements (2 rows apart):

Preference: 100.0% favor same column (effect size: 50.0%)
95% CI: [92.6%, 100.0%]
Statistical test: p = 0.0000 *** (n=48)

SAME COLUMN (0) VS DISTANT COLUMNS (2-3) - ROW CONTROLLED:
(Excludes same-row bigrams per methodology)

Reach Movements (1 row apart):
Preference: 64.6% favor same column (effect size: 14.6%)
95% CI: [56.1%, 72.3%]
Statistical test: p = 0.0009 *** (n=130)

Hurdle Movements (2 rows apart):
Preference: 71.0% favor distant columns (effect size: 21.0%)
95% CI: [58.7%, 80.8%]
Statistical test: p = 0.0010 *** (n=62)

ADJACENT (1) VS DISTANT (2-3) COLUMNS - BY ROW PATTERN:

Same Row Movements (0 row separation):
Preference: 68.5% favor adjacent columns (effect size: 18.5%)
95% CI: [63.3%, 73.2%]
Statistical test: p = 0.0000 *** (n=336)

PROLIFIC DATA

Instances analyzed: 1356

OVERALL COLUMN SEPARATION PREFERENCE (WITH ROW CONTROLS):
Preference rate: 54.3% favor smaller distances (effect size: 4.3%)
95% CI: [51.6%, 56.9%]
Statistical test: p = 0.0016 ** (n=1356)

SAME COLUMN (0) VS ADJACENT COLUMN (1) - ROW CONTROLLED:
(Excludes same-row bigrams per methodology)

Hurdle Movements (2 rows apart):
Preference: 50.9% favor same column (effect size: 0.9%)
95% CI: [46.8%, 54.9%]
Statistical test: p = 0.6795 (n=586)

Reach Movements (1 row apart):
Preference: 53.9% favor same column (effect size: 3.9%)
95% CI: [49.7%, 58.1%]
Statistical test: p = 0.0691 (n=534)

SAME COLUMN (0) VS DISTANT COLUMNS (2-3) - ROW CONTROLLED:
(Excludes same-row bigrams per methodology)

Hurdle Movements (2 rows apart):
  Preference: 58.8% favor same column (effect size: 8.8%)
  95% CI: [50.4%, 66.7%]
  Statistical test: p = 0.0396 * (n=136)

Reach Movements (1 row apart):
  Preference: 63.6% favor same column (effect size: 13.6%)
  95% CI: [48.9%, 76.2%]
  Statistical test: p = 0.0704 (n=44)

ADJACENT (1) VS DISTANT (2-3) COLUMNS - BY ROW PATTERN:

Same Row Movements (0 row separation):
  Preference: 75.0% favor adjacent columns (effect size: 25.0%)
  95% CI: [62.3%, 84.5%]
  Statistical test: p = 0.0002 *** (n=56)

Focus on same-row data

- Weak effect sizes: Most same-column vs adjacent-column comparisons show small effects with poor statistical significance.
- Small sample sizes: Some critical comparisons have tiny samples (n=44, n=136), making them unreliable for setting objective weights.
- One strong signal: The only robust finding is that adjacent columns are strongly preferred over distant columns in same-row movements (18.5% effect size, n=336, p=0.0000; 25.0% effect size, n=56, p<0.001).

```python
# Non-Prolific: n=336, effect=0.185, CI width = 0.099
# Prolific: n=56, effect=0.250, CI width = 0.222
weight1 = 336 / 0.099  # 3394
weight2 = 56 / 0.222  # 252

combined_effect = (0.185 * 3394 + 0.250 * 252) / (3394 + 252)   # 0.189
score_reach = 1 - 0.189  # 0.811
```

(5) Inward/outward rolls

Inward vs outward roll preference (same key pairs, different directions)

Methods:
  - Compares same key pairs in both movement directions
  - Inward roll: Finger number increases (pinky → index)
  - Outward roll: Finger number decreases (index → pinky)
  - Excludes same-column bigrams (no roll motion possible)
  - Controls for key identity, distance, and quality differences

Examples:
- 'as' (inward: finger 1→2) vs 'sa' (outward: finger 2→1)
- 'df' (inward: finger 3→4) vs 'fd' (outward: finger 4→3)
- 'aw' (inward: finger 1→2) vs 'wa' (outward: finger 2→1)

PROLIFIC DATA

Instances analyzed: 856

CONSTRAINED INWARD VS OUTWARD ROLL PREFERENCE:
Inward roll preference rate: 55.8% (effect size: 5.8%)
95% CI: [52.5%, 59.1%]
Statistical test: p = 0.0006 *** (n=856)

Hurdle Movements (2 rows apart):
Examples: 'az'/'za', 'qx'/'xq', 'ec'/'ce'
Preference: 53.2% favor outward roll (effect size: 3.2%)
95% CI: [46.6%, 59.6%]
Statistical test: p = 0.3474 (n=222)

Same Row Movements:
Examples: 'as'/'sa', 'df'/'fd', 'qw'/'wq'
Preference: 72.1% favor inward roll (effect size: 22.1%)
95% CI: [65.7%, 77.8%]
Statistical test: p = 0.0000 *** (n=208)

Reach Movements (1 row apart):
Examples: 'aw'/'wa', 'dr'/'rd', 'sz'/'zs'
Preference: 52.6% favor inward roll (effect size: 2.6%)
95% CI: [47.8%, 57.3%]
Statistical test: p = 0.2865 (n=426)

SAME-ROW SCORING

22.1% effect size → ±0.11 bonus/penalty
1.0 - (22.1% effect size) = 0.779

Consideration: CONTEXT-DEPENDENT TRIGRAM SCORING

```python
# MOO Scoring:
def score_trigram_order(finger1, finger2, finger3, row1, row2, row3):
    if finger1 == finger2 == finger3:    # 1 finger
        return 0    # same finger
    elif finger1 == finger2 or finger2 == finger3:    # 2 fingers
        if key1 == key2 or key2 == key3:
            return 0.5    # repeat key treated the same as a cross-row trigram
        else:
            return 0    # mixed finger patterns
```

```
    else:    # 3 different fingers
        # Same-row trigrams: Apply empirical inward preference
        if row1 == row2 == row3:
            if finger1 < finger2 < finger3:
                return 1.0      # inward roll
            elif finger1 > finger2 > finger3:
                return 0.779     # outward roll (empirical penalty)
        # Cross-row trigrams: Keep equal weighting (no empirical data)
        elif finger1 < finger2 < finger3 or finger1 > finger2 > finger3:
            return 0.5      # both directions neutral
        return 0.0  # mixed/same finger patterns
    if finger1 != finger2 or finger2 != finger3:    # 2 different keys
        return 0.779    # repeat keys
    else:
        return 0    # same finger
```

(6) SIDE REACH

Side reach analysis: Same-row bigrams only (no same-column)

Methods:
- Same-row bigrams only: Both keys on same keyboard row (0 row separation)
- No same-column bigrams: Excludes same-finger movements for cleaner results
- Standard area: Bigrams using only columns 1-4 (Q,W,E,R,A,S,D,F,Z,X,C,V)
- Side reach: Bigrams containing column 5 keys (T,G,B)
- Purest test of side reach cost without movement complexity confounds

Examples:
- Row 1: 'qw' vs 'qt', 'er' vs 'et', 'wr' vs 'wt'
- Row 2: 'as' vs 'ag', 'df' vs 'dg', 'sf' vs 'sg'
- Row 3: 'zx' vs 'zb', 'cv' vs 'cb', 'xv' vs 'xb'

Exclusions:
- Different rows: 'aw' vs 'at' (reach movements)
- Same column: 'de' vs 'gt' (same finger movements)

PROLIFIC DATA

Instances analyzed: 804

SAME-ROW SIDE REACH PREFERENCE:
Standard area preference rate: 65.4% (effect size: 15.4%)
95% CI: [62.1%, 68.6%]
Statistical test: p = 0.0000 *** (n=804)

SCORING
side_reach_score: 1 - 0.154 = 0.846

Compound penalty logic (each penalty reduces the remaining quality rather than the total):
- 1 side reach: retains 84.6% quality
- 2 side reaches: retains 84.6% × 84.6% = 71.6% quality
- This models diminishing marginal cost appropriately
- An 15.4% effect size can be interpreted as "side reach bigrams retain 84.6% of standard quality" rather than "side reach costs exactly 15.4% points."

Bigram Scoring System

1. Key preference score (empirical Bradley-Terry tiers)

```
tier_values = {
    'F': 1.000,
    'D': 0.870,
    'E': 0.646,
    'S': 0.646,
    'V': 0.568,
    'R': 0.568,
    'W': 0.472,
    'A': 0.410,
    'C': 0.410,
    'Z': 0.137,
    'Q': 0.137,
    'X': 0.137
}

key_score = 0
for key in [char1, char2]:
    key_score += tier_values.get(key, 0)  # Get tier value or 0 if not found

scores['position'] = key_score / 2.0  # Average over 2 keys
```

2. Row separation score (empirical meta-analysis of left-hand bigrams)

```
#    1.000: 2 keys in the same row
#    0.588: 2 keys in adjacent rows (reach)
#    0.000: 2 keys straddling home row (hurdle)
if hand1 != hand2:
    scores['rows'] = 1.0         # Opposite hands
else:
    if row_gap == 0:
        scores['rows'] = 1.0     # Same-row
    elif row_gap == 1:
        scores['rows'] = 0.588  # Adjacent row (reach)
    else:
```

```
        scores['rows'] = 0.0    # Hurdle
```

3. Column separation (same-row) score (empirical meta-analysis of left-hand bigrams)

```
#    (empirical meta-analysis of left-hand bigrams)
#    1.00:  adjacent columns in the same row (or 2 hands)
#    0.811: remote columns in the same row
#    0.50:  other
if hand1 != hand2:
    scores['columns'] = 1.0    # High score for opposite hands
elif column_gap == 1 and row_gap == 0:
    scores['columns'] = 1.0    # Adjacent same-row (baseline)
elif column_gap >= 2 and row_gap == 0:
    scores['columns'] = 0.811    # Distant same-row (empirical penalty)
else:
    scores['columns'] = 0.5    # Neutral score for everything else
```

4. Roll direction (same-row) score (empirical analysis of left-hand bigrams)

```
#    1.000: same-row inward roll (or 2 hands)
#    0.779: same-row outward roll
#    0.500: other
#    0.000: same finger
scores['order'] = 0.5    # Neutral score by default
if hand1 != hand2:
    scores['order'] = 1.0    # Opposite hands
elif finger1 == finger2:
    scores['order'] = 0.0    # Same finger
elif (hand1 == hand2 and finger1 != finger2 and row_gap == 0):
    if finger2 > finger1:    # Same-row inward roll (pinky → index)
        scores['order'] = 1.0
    elif finger2 < finger1:    # Same-row outward roll (index → pinky)
        scores['order'] = 0.779    # 100 - 22.1% effect penalty
```

5. Side reach multiplicative penalty (empirical analysis of left-hand bigrams)

```
#    1.000: 0 column 5 keys
#    0.846: 1 column 5 key
#    0.716: 2 column 5 keys
column_5_keys = {'T', 'G', 'B', 'Y', 'H', 'N'}
scores['side'] = 1.0
for key in [char1, char2]:
    if key.upper() in column_5_keys:
        scores['side'] *= 0.846    # Apply 15.4% penalty each time
```