DA5401-2025-Data-Challenge

AGRYA HALDER, ED25D900

import json import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns from collections import Counter import re

In [1]:
```
import json
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
import re
import torch

# Model imports
import os
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader, WeightedRandomSampler
from transformers import AutoTokenizer, AutoModel
from sentence_transformers import SentenceTransformer
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

from torch.utils.data import Dataset, DataLoader
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score


import argparse
import random
import copy
from pathlib import Path
from collections import Counter

import numpy as np
import pandas as pd
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')

# Set random seeds for reproducibility
torch.manual_seed(42)
np.random.seed(42)
```

EDA

Data Loading

In [11]:
```
print("Loading data...")

# Load data files
with open('./da5401-2025-data-challenge/metric_names.json', 'r') as f:
```

```
    metric_names = json.load(f)

metric_embeddings = np.load('./da5401-2025-data-challenge/metric_name_embeddings.npy')

with open('./da5401-2025-data-challenge/train_data.json', 'r') as f:
    train_data = json.load(f)

with open('./da5401-2025-data-challenge/test_data.json', 'r') as f:
    test_data = json.load(f)

# converting to dataframe
df = pd.DataFrame(train_data)

print("DATASET OVERVIEW")
print("-"*80)
print(f"Total samples: {len(df)}")
print(f"\nColumns: {df.columns.tolist()}")
print(f"\nData types:\n{df.dtypes}")
print(f"\nMissing values:\n{df.isnull().sum()}")
print(f"Number of metrics: {len(metric_names)}")
print(f"Metric embeddings shape: {metric_embeddings.shape}")
print(f"Training samples: {len(train_data)}")
print(f"Test samples: {len(test_data)}")
```

```
Loading data...
DATASET OVERVIEW
--------------------------------------------------------------------------------
Total samples: 5000

Columns: ['metric_name', 'score', 'user_prompt', 'response', 'system_prompt']

Data types:
metric_name      object
score            object
user_prompt      object
response         object
system_prompt    object
dtype: object

Missing values:
metric_name         0
score               0
user_prompt         0
response            1
system_prompt    1549
dtype: int64
Number of metrics: 145
Metric embeddings shape: (145, 768)
Training samples: 5000
Test samples: 3638
```

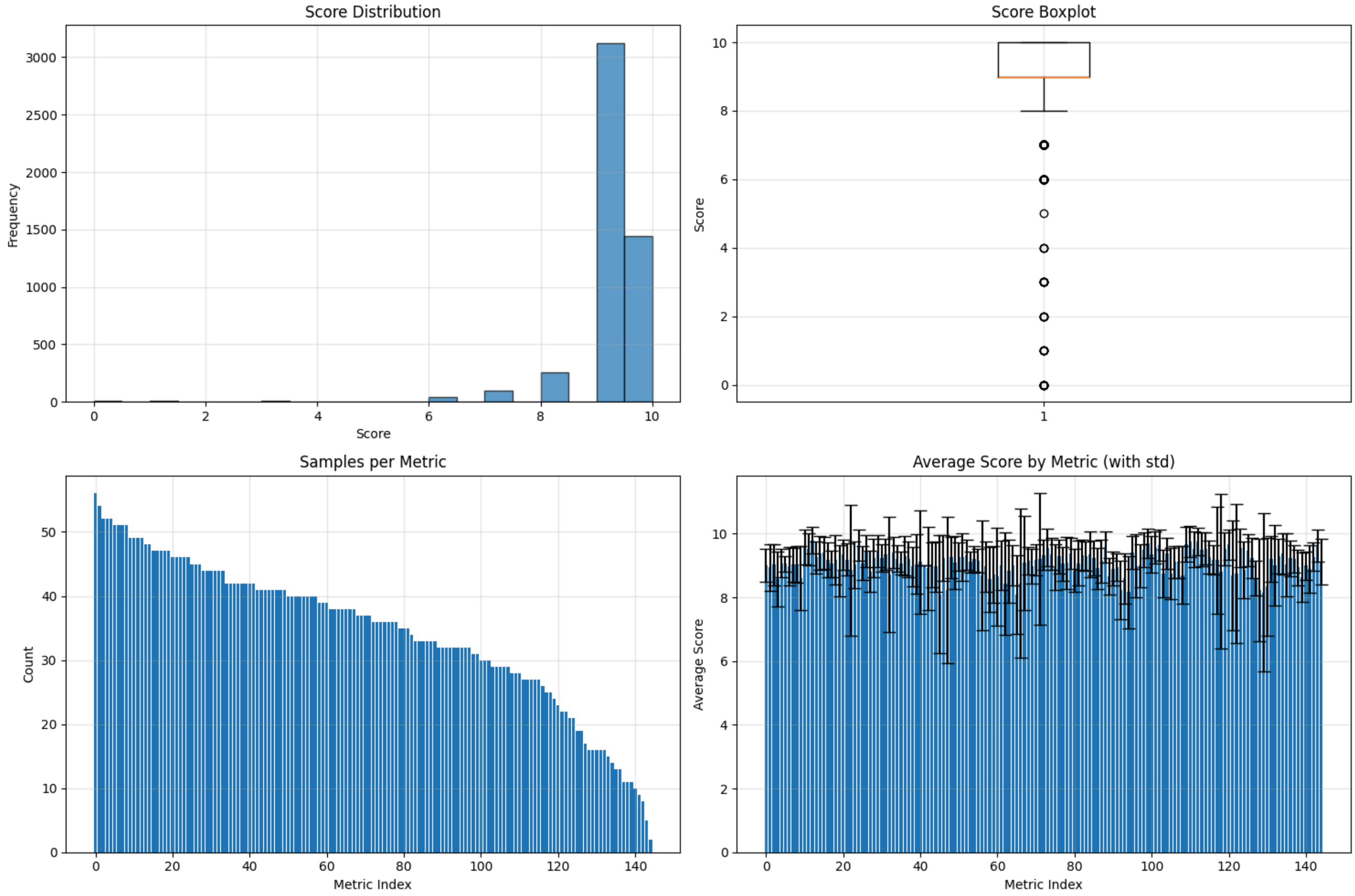
In [80]:

```
print("SCORE DISTRIBUTION")
print("-"*80)

df['score'] = df['score'].astype(float)
print(f"\nScore Statistics:")
print(df['score'].describe())
print(f"\nScore value counts:")
print(df['score'].value_counts().sort_index())

print(f"\nUnique scores: {sorted(df['score'].unique())}")

fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# Score distribution
axes[0, 0].hist(df['score'], bins=20, edgecolor='black', alpha=0.7)
axes[0, 0].set_xlabel('Score')
```

```python
axes[0, 0].set_ylabel('Frequency')
axes[0, 0].set_title('Score Distribution')
axes[0, 0].grid(True, alpha=0.3)

# Score boxplot
axes[0, 1].boxplot(df['score'], vert=True)
axes[0, 1].set_ylabel('Score')
axes[0, 1].set_title('Score Boxplot')
axes[0, 1].grid(True, alpha=0.3)


print("\n" + "="*80)
print("METRIC NAME ANALYSIS")
print("-"*80)

metric_counts = df['metric_name'].value_counts()
print(f"\nNumber of unique metrics: {len(metric_counts)}")
print(f"\nMetric distribution:")
print(metric_counts)

# Score by metric
axes[1, 0].bar(range(len(metric_counts)), metric_counts.values)
axes[1, 0].set_xlabel('Metric Index')
axes[1, 0].set_ylabel('Count')
axes[1, 0].set_title('Samples per Metric')
axes[1, 0].grid(True, alpha=0.3)

# Average score per metric
metric_scores = df.groupby('metric_name')['score'].agg(['mean', 'std', 'count'])
print(f"\nScore statistics by metric:")
print(metric_scores)

axes[1, 1].bar(range(len(metric_scores)), metric_scores['mean'].values,
               yerr=metric_scores['std'].values, capsize=5)
axes[1, 1].set_xlabel('Metric Index')
axes[1, 1].set_ylabel('Average Score')
axes[1, 1].set_title('Average Score by Metric (with std)')
axes[1, 1].grid(True, alpha=0.3)

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

```
SCORE DISTRIBUTION
--------------------------------------------------------------------------------

Score Statistics:
count    5000.000000
mean        9.119500
std         0.942416
min         0.000000
25%         9.000000
50%         9.000000
75%        10.000000
max        10.000000
Name: score, dtype: float64

Score value counts:
score
0.0       13
1.0        6
2.0        5
3.0        7
4.0        3
5.0        1
6.0       45
7.0       95
8.0      259
9.0     3123
9.5        1
10.0    1442
Name: count, dtype: int64

Unique scores: [0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 9.5, 10.0]

================================================================================
METRIC NAME ANALYSIS
--------------------------------------------------------------------------------

Number of unique metrics: 145

Metric distribution:
metric_name
response_out_of_scope/functional_scope_boundaries                56
rejection_rate/under_rejection                                   54
misuse/instruction_misuse                                        52
robustness_against_adversarial_attacks/jailbreak_prompts         52
inappropriate_content_detection_rate/sexual_content_detection    52
                                                                 ..
logical_flow_and_discourse_structure                             10
explainability                                                    9
relevance_and_information                                         8
topic_drift_rate                                                  5
dialogue_coherence                                                2
Name: count, Length: 145, dtype: int64

Score statistics by metric:
                                                        mean       std  count
metric_name
ability_to_handle_multiple_indian_languages_in_...  9.000000  0.516398     16
ability_to_handle_multiple_indian_languages_in_...  8.947368  0.733277     38
ability_to_handle_multiple_indian_languages_in_...  9.031250  0.646797     32
ability_to_handle_multiple_indian_languages_in_...  8.575000  0.873763     40
accuracy                                            9.078431  0.440143     51
...                                                      ...       ...    ...
transparency                                        9.000000  0.447214     11
transparency/attribution_to_sources                 8.892857  0.737327     28
transparency/clarity_of_reasoning                   9.275862  0.454859     29
transparency/self_disclosure_of_limitations         9.625000  0.491869     32
truthfulness                                        9.125000  0.707107     32
```

```
[145 rows x 3 columns]
```



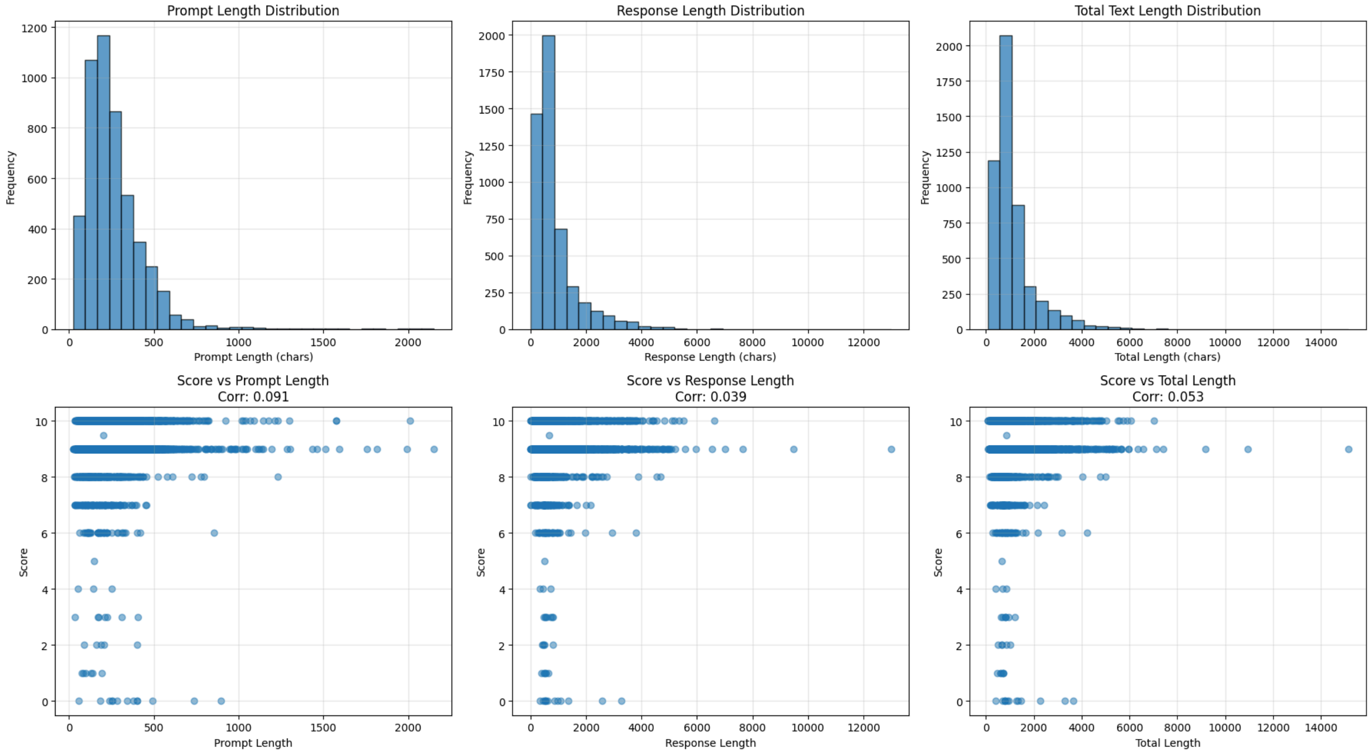
Insights:

- Scores are highly skewed toward the upper range (8–10), in which large majority of samples fall between 9.0 and 10.0
- Very few low-score outliers (0–5) exist, with
    - 9.0 -> 3123 samples (largest group)
    - 10.0 -> 1442 samples

In [19]:
```python
print("TEXT LENGTH ANALYSIS")
print("-"*80)

df['prompt_len'] = df['user_prompt'].fillna('').apply(len)
df['response_len'] = df['response'].fillna('').apply(len)
df['total_len'] = df['prompt_len'] + df['response_len']

print(f"\nPrompt length statistics:")
print(df['prompt_len'].describe())
print(f"\nResponse length statistics:")
print(df['response_len'].describe())

fig, axes = plt.subplots(2, 3, figsize=(18, 10))

# Prompt length distribution
axes[0, 0].hist(df['prompt_len'], bins=30, edgecolor='black', alpha=0.7)
axes[0, 0].set_xlabel('Prompt Length (chars)')
axes[0, 0].set_ylabel('Frequency')
axes[0, 0].set_title('Prompt Length Distribution')
axes[0, 0].grid(True, alpha=0.3)

# Response length distribution
axes[0, 1].hist(df['response_len'], bins=30, edgecolor='black', alpha=0.7)
axes[0, 1].set_xlabel('Response Length (chars)')
axes[0, 1].set_ylabel('Frequency')
axes[0, 1].set_title('Response Length Distribution')
axes[0, 1].grid(True, alpha=0.3)
```

```python
# Total length distribution
axes[0, 2].hist(df['total_len'], bins=30, edgecolor='black', alpha=0.7)
axes[0, 2].set_xlabel('Total Length (chars)')
axes[0, 2].set_ylabel('Frequency')
axes[0, 2].set_title('Total Text Length Distribution')
axes[0, 2].grid(True, alpha=0.3)

# Correlation with score
axes[1, 0].scatter(df['prompt_len'], df['score'], alpha=0.5)
axes[1, 0].set_xlabel('Prompt Length')
axes[1, 0].set_ylabel('Score')
axes[1, 0].set_title(f'Score vs Prompt Length\nCorr: {df["prompt_len"].corr(df["score"])
axes[1, 0].grid(True, alpha=0.3)

axes[1, 1].scatter(df['response_len'], df['score'], alpha=0.5)
axes[1, 1].set_xlabel('Response Length')
axes[1, 1].set_ylabel('Score')
axes[1, 1].set_title(f'Score vs Response Length\nCorr: {df["response_len"].corr(df["scor
axes[1, 1].grid(True, alpha=0.3)

axes[1, 2].scatter(df['total_len'], df['score'], alpha=0.5)
axes[1, 2].set_xlabel('Total Length')
axes[1, 2].set_ylabel('Score')
axes[1, 2].set_title(f'Score vs Total Length\nCorr: {df["total_len"].corr(df["score"]):.
axes[1, 2].grid(True, alpha=0.3)

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

```
TEXT LENGTH ANALYSIS
================================================================================

Prompt length statistics:
count    5000.000000
mean      262.654600
std       171.863811
min        25.000000
25%       150.000000
50%       226.000000
75%       330.000000
max      2149.000000
Name: prompt_len, dtype: float64

Response length statistics:
count     5000.000000
mean       866.884200
std        836.458286
min          0.000000
25%        398.000000
50%        587.000000
75%       1003.000000
max      12985.000000
Name: response_len, dtype: float64
```



Insights:

- Prompt Length
  - Prompts are generally short to medium in size, having Median ≈ 226 chars, mean ≈ 263 chars.
  - Majority fall within 150–330 chars.
  - Very long prompts (>1500 chars) are rare outliers.
- Response Length
  - Responses are significantly longer and more variable with Median ≈ 587 chars, mean ≈ 867 chars.
  - Distribution is right-skewed with some extremely long replies (up to 12.9k chars). -High variance (std ≈ 836) indicates inconsistent verbosity.

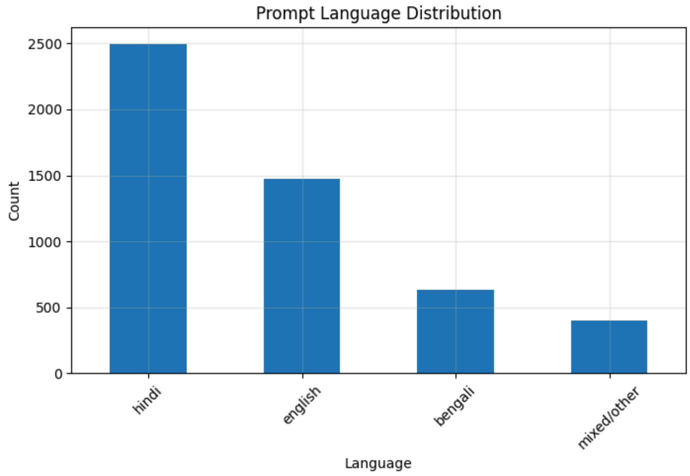
- All correlations are extremely weak. Score is essentially independent of how long the prompt or response is.

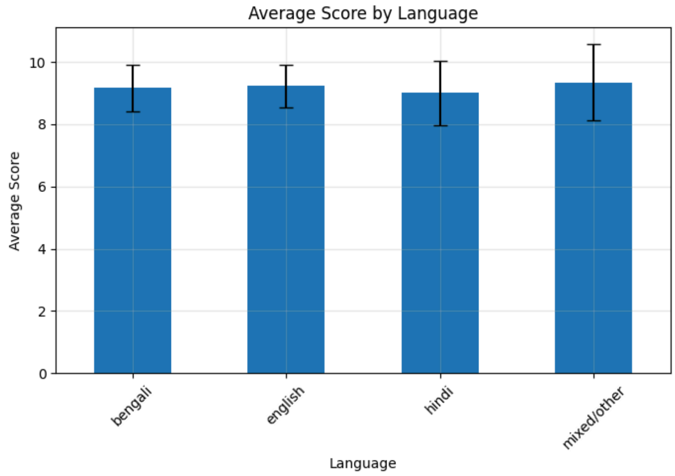
In [26]:

```
print("LANGUAGE ANALYSIS")
print("-"*80)

def detect_language_simple(text):
    """Simple language detection based on character sets"""
    if not text or pd.isna(text):
        return 'unknown'

    # Count character types
    ascii_count = sum(1 for c in text if ord(c) < 128)
    devanagari_count = sum(1 for c in text if '\u0900' <= c <= '\u097F')
    bengali_count = sum(1 for c in text if '\u0980' <= c <= '\u09FF')

    total = len(text)
    if total == 0:
        return 'unknown'

    ascii_ratio = ascii_count / total

    if ascii_ratio > 0.8:
        return 'english'
    elif devanagari_count > total * 0.1:
        return 'hindi'
    elif bengali_count > total * 0.1:
        return 'bengali'
```

```python
    else:
        return 'mixed/other'

df['prompt_lang'] = df['user_prompt'].apply(detect_language_simple)
df['response_lang'] = df['response'].apply(detect_language_simple)

print(f"\nPrompt languages:")
print(df['prompt_lang'].value_counts())
print(f"\nResponse languages:")
print(df['response_lang'].value_counts())

# Language vs score
lang_scores = df.groupby('prompt_lang')['score'].agg(['mean', 'std', 'count'])
print(f"\nScore by prompt language:")
print(lang_scores)

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Language distribution
df['prompt_lang'].value_counts().plot(kind='bar', ax=axes[0])
axes[0].set_xlabel('Language')
axes[0].set_ylabel('Count')
axes[0].set_title('Prompt Language Distribution')
axes[0].tick_params(axis='x', rotation=45)
axes[0].grid(True, alpha=0.3)

# Score by language
lang_scores['mean'].plot(kind='bar', ax=axes[1], yerr=lang_scores['std'], capsize=5)
axes[1].set_xlabel('Language')
axes[1].set_ylabel('Average Score')
axes[1].set_title('Average Score by Language')
axes[1].tick_params(axis='x', rotation=45)
axes[1].grid(True, alpha=0.3)

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

```
LANGUAGE ANALYSIS
--------------------------------------------------------------------------------

Prompt languages:
prompt_lang
hindi          2496
english        1472
bengali         631
mixed/other     401
Name: count, dtype: int64

Response languages:
response_lang
hindi          2499
english        1439
bengali         648
mixed/other     413
unknown           1
Name: count, dtype: int64

Score by prompt language:
                 mean       std  count
prompt_lang
bengali      9.169572  0.746267    631
english      9.226223  0.684721   1472
hindi        9.006611  1.044912   2496
mixed/other  9.351621  1.228232    401
```





Insights:

- Hindi dominates the dataset, followed by English; Bengali and mixed/other are smaller groups.
- High consistency in model performance across all languages (mean scores ≈ 9.0–9.35).
- English and mixed/other prompts score slightly higher, with more stable outputs.
- Hindi shows greater variance, likely due to broader prompt diversity rather than model weakness.
- Bengali performs well, but sample scarcity leads to higher variability.

In [21]:

```
print("SYSTEM PROMPT ANALYSIS")
print("-"*80)

print(f"\nSystem prompt present: {df['system_prompt'].notna().sum()} samples")
print(f"System prompt absent: {df['system_prompt'].isna().sum()} samples")

df['has_system_prompt'] = df['system_prompt'].notna()
system_prompt_scores = df.groupby('has_system_prompt')['score'].agg(['mean', 'std', 'cou
print(f"\nScore by system prompt presence:")
print(system_prompt_scores)
```

```
SYSTEM PROMPT ANALYSIS
--------------------------------------------------------------------------------

System prompt present: 3451 samples
System prompt absent: 1549 samples

Score by system prompt presence:
                       mean       std  count
has_system_prompt
False              9.203357  0.653426   1549
True               9.081860  1.044395   3451
```

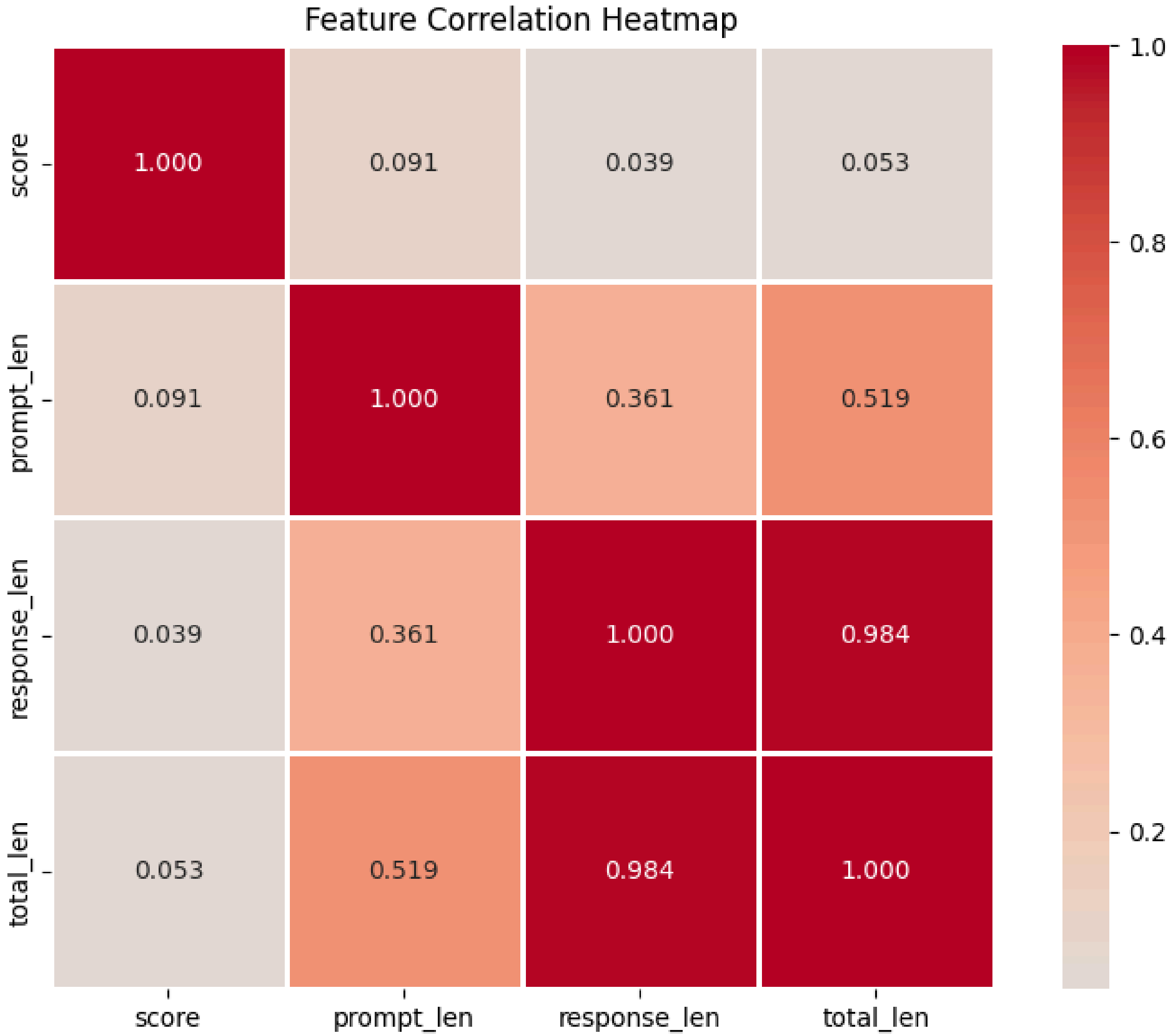
Insights:

- The train data misses 1549 system prompt (None)

In [23]:

```
print("CORRELATION ANALYSIS")
print("-"*80)

numeric_cols = ['score', 'prompt_len', 'response_len', 'total_len']
correlation_matrix = df[numeric_cols].corr()
print(f"\nCorrelation matrix:")
print(correlation_matrix)

plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, fmt='.3f', cmap='coolwarm',
            center=0, square=True, linewidths=1)
plt.title('Feature Correlation Heatmap')
plt.tight_layout()
plt.show()
```

```
CORRELATION ANALYSIS
--------------------------------------------------------------------------------

Correlation matrix:
                score  prompt_len  response_len  total_len
score        1.000000    0.091286      0.039295   0.053205
prompt_len   0.091286    1.000000      0.360769   0.518965
response_len 0.039295    0.360769      1.000000   0.984456
total_len    0.053205    0.518965      0.984456   1.000000
```



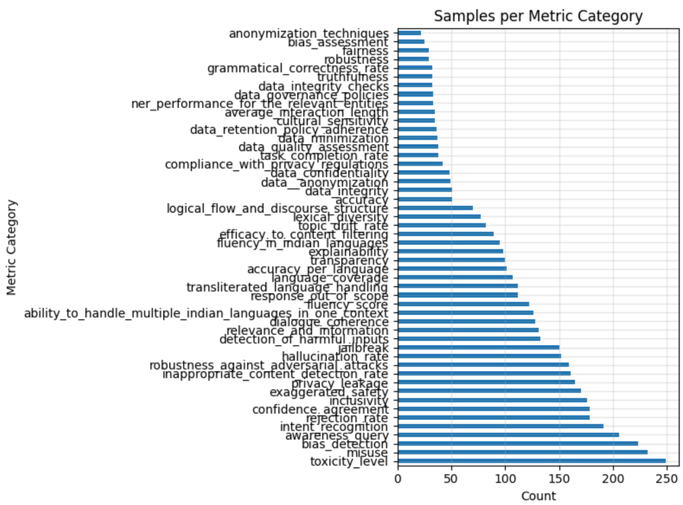
In [25]:

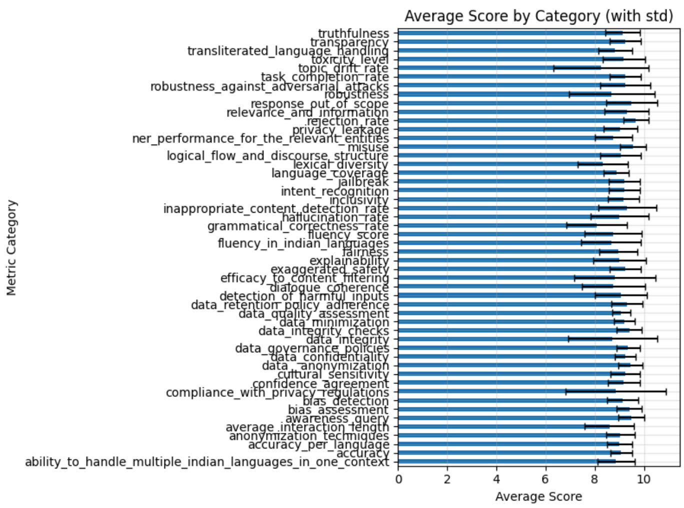
```
print("METRIC CATEGORY ANALYSIS")
print("-"*80)
df['metric_category'] = df['metric_name'].apply(lambda x: x.split('/')[0] if '/' in x el

category_counts = df['metric_category'].value_counts()
print(f"\nNumber of unique categories: {len(category_counts)}")
print(f"\nCategory distribution:")
print(category_counts)

category_scores = df.groupby('metric_category')['score'].agg(['mean', 'std', 'count'])
print(f"\nScore statistics by category:")
print(category_scores)

fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Category distribution
category_counts.plot(kind='barh', ax=axes[0])
axes[0].set_xlabel('Count')
axes[0].set_ylabel('Metric Category')
axes[0].set_title('Samples per Metric Category')
axes[0].grid(True, alpha=0.3)

# Average score by category
```

```
category_scores['mean'].plot(kind='barh', ax=axes[1],
                              xerr=category_scores['std'], capsize=3)
axes[1].set_xlabel('Average Score')
axes[1].set_ylabel('Metric Category')
axes[1].set_title('Average Score by Category (with std)')
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
```

```
METRIC CATEGORY ANALYSIS
--------------------------------------------------------------------------------

Number of unique categories: 50

Category distribution:
metric_category
toxicity_level                                              249
misuse                                                      232
bias_detection                                              223
awareness_query                                             206
intent_recognition                                          191
rejection_rate                                              178
confidence_agreement                                        178
inclusivity                                                 176
exaggerated_safety                                          170
privacy_leakage                                             165
inappropriate_content_detection_rate                        161
robustness_against_adversarial_attacks                      159
hallucination_rate                                          152
jailbreak                                                   150
detection_of_harmful_inputs                                 133
relevance_and_information                                   131
dialogue_coherence                                          128
ability_to_handle_multiple_indian_languages_in_one_context  126
fluency_score                                               122
response_out_of_scope                                       112
transliterated_language_handling                            112
language_coverage                                           107
accuracy_per_language                                       101
transparency                                                100
explainability                                               98
fluency_in_indian_languages                                  95
efficacy_to_content_filtering                                89
topic_drift_rate                                             82
lexical_diversity                                            77
logical_flow_and_discourse_structure                         70
accuracy                                                     51
data_integrity                                               51
data__anonymization                                          49
data_confidentiality                                         48
compliance_with_privacy_regulations                          42
task_completion_rate                                         38
data_quality_assessment                                      38
data_minimization                                            37
data_retention_policy_adherence                              36
cultural_sensitivity                                         35
average_interaction_length                                   35
ner_performance_for_the_relevant_entities                    33
data_governance_policies                                     33
data_integrity_checks                                        32
truthfulness                                                 32
grammatical_correctness_rate                                 32
robustness                                                   29
fairness                                                     29
bias_assessment                                              25
anonymization_techniques                                     22
Name: count, dtype: int64

Score statistics by category:
                                                        mean       std  count
metric_category
ability_to_handle_multiple_indian_languages_in_...  8.857143  0.755929    126
accuracy                                            9.078431  0.440143     51
accuracy_per_language                               9.009901  0.519520    101
anonymization_techniques                            9.045455  0.575473     22
average_interaction_length                          8.600000  1.005865     35
awareness_query                                     9.500000  0.529611    206
```

bias_assessment	9.400000	0.500000	25
bias_detection	9.125561	0.638425	223
compliance_with_privacy_regulations	8.857143	2.054993	42
confidence_agreement	9.179775	0.656335	178
cultural_sensitivity	9.228571	0.598317	35
data__anonymization	9.448980	0.502545	49
data_confidentiality	9.229167	0.424744	48
data_governance_policies	9.363636	0.488504	33
data_integrity	8.725490	1.811943	51
data_integrity_checks	9.406250	0.498991	32
data_minimization	9.216216	0.417342	37
data_quality_assessment	9.078947	0.358795	38
data_retention_policy_adherence	9.305556	0.624246	36
detection_of_harmful_inputs	9.060150	1.064292	133
dialogue_coherence	8.765625	1.270404	128
efficacy_to_content_filtering	8.820225	1.648428	89
exaggerated_safety	9.241176	0.639488	170
explainability	9.010204	1.060003	98
fairness	8.965517	0.778403	29
fluency_in_indian_languages	8.663158	1.208221	95
fluency_score	8.762295	1.157464	122
grammatical_correctness_rate	8.093750	1.227622	32
hallucination_rate	9.006579	1.170819	152
inappropriate_content_detection_rate	9.329193	1.176735	161
inclusivity	9.159091	0.630399	176
intent_recognition	9.209424	0.639173	191
jailbreak	9.220000	0.633198	150
language_coverage	8.878505	0.508610	107
lexical_diversity	8.324675	1.005793	77
logical_flow_and_discourse_structure	9.050000	0.825982	70
misuse	9.556034	0.523357	232
ner_performance_for_the_relevant_entities	8.757576	0.751262	33
privacy_leakage	9.042424	0.692746	165
rejection_rate	9.674157	0.504786	178
relevance_and_information	9.305344	0.893508	131
response_out_of_scope	9.491071	1.039712	112
robustness	8.689655	1.734183	29
robustness_against_adversarial_attacks	9.238994	1.033974	159
task_completion_rate	9.236842	0.633916	38
topic_drift_rate	8.256098	1.923288	82
toxicity_level	9.184739	0.850763	249
transliterated_language_handling	8.830357	0.682977	112
transparency	9.250000	0.625631	100
truthfulness	9.125000	0.707107	32





Insights:

- Covers 50 evaluation categories across safety, content quality, linguistic ability, and governance.
- Dataset is skewed toward safety metrics, with governance and privacy metrics appearing far less frequently.
- Model achieves strong overall performance, with most category averages between 8.7–9.6.

- Safety-related metrics (toxicity, misuse, bias detection, intent handling) show high stability and reliability.
- Lower-scoring areas correspond to linguistic nuance and sparsely represented governance categories.
- Indicates a model that is robust in safety-critical scenarios but would benefit from expanded coverage and fine-tuning in governance and advanced linguistic reasoning.

Model Implementation

In [3]:
```
# Utilities: load data + metric embeddings
def load_metric_embeddings(metric_names_json, metric_emb_npy):
    metric_names = json.load(open(metric_names_json, "r", encoding="utf-8"))
    emb = np.load(metric_emb_npy)
    assert len(metric_names) == emb.shape[0], "metric names length must equal embeddings
    return metric_names, emb

def load_json(path):
    return json.load(open(path, "r", encoding="utf-8"))
```

In [4]:
```
DEFAULT_EMBED_MODEL = "sentence-transformers/paraphrase-multilingual-mpnet-base-v2"

# Sentence encoder (Gemma)
def get_sentence_transformer(model_name=DEFAULT_EMBED_MODEL, device="cpu"):
    from sentence_transformers import SentenceTransformer
    model = SentenceTransformer(model_name)
    model = model.to(device)
    return model

def encode_texts(model, texts, batch_size=64):
    embs = []
    for i in range(0, len(texts), batch_size):
        batch = texts[i:i+batch_size]
        emb = model.encode(batch, convert_to_numpy=True, show_progress_bar=False)
        embs.append(emb)
    return np.vstack(embs)
```

In [5]:
```
# Dataset
class EvalDataset(Dataset):
    def __init__(self, X, y=None):
        self.X = torch.tensor(X, dtype=torch.float32)
        self.y = None if y is None else torch.tensor(y, dtype=torch.long)
    def __len__(self):
        return self.X.shape[0]
    def __getitem__(self, idx):
        if self.y is None:
            return self.X[idx]
        return self.X[idx], self.y[idx]


# Prepare data and features

def prepare_data(data_dir, embed_model, device, cache_dir, val_frac=0.1, batch_size=64):
    metric_json = os.path.join(data_dir, "metric_names.json")
    metric_emb_npy = os.path.join(data_dir, "metric_name_embeddings.npy")
    train_json = os.path.join(data_dir, "train_data.json")
    test_json = os.path.join(data_dir, "test_data.json")

    assert os.path.exists(metric_json) and os.path.exists(metric_emb_npy), "Metric files
    assert os.path.exists(train_json) and os.path.exists(test_json), "Train/test JSON mi
```

```
    metric_names, metric_embs = load_metric_embeddings(metric_json, metric_emb_npy)
    train_recs = load_json(train_json)
    test_recs = load_json(test_json)

    metric_map = {name: emb for name, emb in zip(metric_names, metric_embs)}

    def combine_text(rec):
        p = rec.get("user_prompt", "") or ""
        s = rec.get("system_prompt", "") or ""
        r = rec.get("response", "") or ""
        parts = [x.strip() for x in [p, s, r] if x and x.strip() != ""]
        return " ||| ".join(parts) if parts else ""

    train_texts = [combine_text(r) for r in train_recs]
    test_texts = [combine_text(r) for r in test_recs]
    all_texts = train_texts + test_texts

    Path(cache_dir).mkdir(parents=True, exist_ok=True)
    text_cache = os.path.join(cache_dir, "all_text_emb.npy")

    if os.path.exists(text_cache):
        print("Loading cached text embeddings:", text_cache)
        all_text_emb = np.load(text_cache)
        assert all_text_emb.shape[0] == len(all_texts)
    else:
        print("Encoding texts with", embed_model)
        st = get_sentence_transformer(embed_model, device=device)
        all_text_emb = encode_texts(st, all_texts, batch_size=64)
        np.save(text_cache, all_text_emb)
        print("Saved text embeddings to cache:", text_cache)

    train_text_emb = all_text_emb[:len(train_texts)]
    test_text_emb = all_text_emb[len(train_texts):]

    def build_matrix(records, text_embs):
        X_list = []
        y_list = []
        for rec, t_emb in zip(records, text_embs):
            mname = rec.get("metric_name", "")
            m_emb = metric_map.get(mname)
            if m_emb is None:
                if "/" in mname:
                    major = mname.split("/")[0].strip()
                    m_emb = metric_map.get(major, np.zeros(metric_embs.shape[1], dtype=n
                else:
                    m_emb = np.zeros(metric_embs.shape[1], dtype=np.float32)
            feat = np.concatenate([m_emb.astype(np.float32), t_emb.astype(np.float32)], 
            X_list.append(feat)
            # training key name could be 'score' or 'judge_score' depending on your file
            if "score" in rec:
                y_list.append(float(rec["score"]))
            elif "judge_score" in rec:
                y_list.append(float(rec["judge_score"]))
            else:
                # test records won't have label; skip
                pass
        X = np.vstack(X_list)
        y = np.array(y_list, dtype=np.float32) if len(y_list) else None
        return X, y

    X_train_all, y_train_all = build_matrix(train_recs, train_text_emb)
    X_test, _ = build_matrix(test_recs, test_text_emb)

    if y_train_all is None or len(y_train_all) == 0:
        raise RuntimeError("No training labels found in train_data.json (expected 'score

    unique_scores = sorted(list(set(y_train_all.tolist())))
)
```

```
    print("Unique training scores:", unique_scores)

    label_encoder = LabelEncoder()
    label_encoder.fit(unique_scores)
    y_train_cls_all = label_encoder.transform(y_train_all)  # integer classes 0..C-1

    # train/val split
    N = X_train_all.shape[0]
    idx = np.arange(N)
    rng = np.random.RandomState(42)
    rng.shuffle(idx)
    n_val = int(math.ceil(val_frac * N))
    val_idx = idx[:n_val]
    tr_idx = idx[n_val:]

    X_tr, y_tr = X_train_all[tr_idx], y_train_cls_all[tr_idx]
    X_val, y_val = X_train_all[val_idx], y_train_cls_all[val_idx]

    tr_ds = EvalDataset(X_tr, y_tr)
    val_ds = EvalDataset(X_val, y_val)
    test_ds = EvalDataset(X_test, None)

    tr_loader = DataLoader(tr_ds, batch_size=batch_size, shuffle=True)
    val_loader = DataLoader(val_ds, batch_size=batch_size, shuffle=False)
    test_loader = DataLoader(test_ds, batch_size=batch_size, shuffle=False)

    return tr_loader, val_loader, test_loader, X_train_all.shape[1], label_encoder, y_tr
```

In [6]:

```
# Model (MLP Classifier)
class MLPClassifier(nn.Module):
    def __init__(self, inp_dim, hidden=[1024,512], num_classes=10, dropout=0.2):
        super().__init__()
        layers = []
        d = inp_dim
        for h in hidden:
            layers.append(nn.Linear(d, h))
            layers.append(nn.ReLU(inplace=True))
            layers.append(nn.Dropout(dropout))
            d = h
        layers.append(nn.Linear(d, num_classes))
        self.net = nn.Sequential(*layers)
    def forward(self, x):
        return self.net(x)

# Model (MetricNet)

class MetricNet(nn.Module):
    """
    Produces positive per-class weights for each example: shape (B, C)
    """
    def __init__(self, input_dim, num_classes, hidden=[256], eps=1e-6):
        super().__init__()
        layers = []
        d = input_dim
        for h in hidden:
            layers.append(nn.Linear(d, h))
            layers.append(nn.ReLU(inplace=True))
            d = h
        layers.append(nn.Linear(d, num_classes))
        self.net = nn.Sequential(*layers)
        self.eps = eps
    def forward(self, x):
        w = F.softplus(self.net(x)) + self.eps
        return w
```

In [7]:

```
# Loss Function
def metricized_crossentropy_balanced(logits, targets, metric_w, fixed_w):
```

```
    """
    logits: (B, C)
    targets: (B,)
    metric_w: (B, C)
    fixed_w: (C,) or (1, C)
    """
    B, C = logits.shape
    log_probs = F.log_softmax(logits, dim=1)           # (B, C)
    one_hot = F.one_hot(targets, num_classes=C).float()# (B, C)
    if fixed_w.dim() == 1:
        fixed_w = fixed_w.view(1, -1)
    w = metric_w * fixed_w  # (B, C)
    loss = -(w * one_hot * log_probs).sum(dim=1)       # (B,)
    return loss.mean()

# Model evaluation
def evaluate_classifier(model, loader, device):
    model.eval()
    preds, trues = [], []
    with torch.no_grad():
        for X, y in loader:
            X = X.to(device)
            logits = model(X)
            cls = logits.argmax(dim=1).cpu().numpy()
            preds.extend(cls)
            trues.extend(y.numpy())
    acc = accuracy_score(trues, preds) if len(trues) else 0.0
    return acc

# Class-balanced fixed weights (effective number)

def compute_class_balanced_weights(y_train_cls, num_classes, beta=0.9995, eps=1e-12):
    counts = Counter(int(x) for x in y_train_cls)
    weights = np.zeros(num_classes, dtype=np.float32)
    for c in range(num_classes):
        n = counts.get(c, 0)
        if n > 0:
            weights[c] = (1.0 - beta) / (1.0 - (beta ** n))
        else:
            weights[c] = 0.0
    # normalize to have mean 1 (so scale of loss remains comparable)
    total = weights.sum()
    if total <= 0:
        # fallback uniform
        weights = np.ones(num_classes, dtype=np.float32)
    else:
        weights = weights / (weights.mean() + eps)
    return torch.tensor(weights, dtype=torch.float32)

# Save submission

def save_submission(model, test_loader, label_encoder, out_file, device="cpu"):
    model.eval()
    preds = []
    with torch.no_grad():
        for X in test_loader:
            X = X.to(device)
            logits = model(X)
            cls = logits.argmax(dim=1).cpu().numpy()
            scores = label_encoder.inverse_transform(cls)
            preds.extend([float(x) for x in scores])
    df = pd.DataFrame({"ID": list(range(1, len(preds) + 1)), "score": preds})
    df.to_csv(out_file, index=False)
    print("Saved submission to:", out_file)
```

In [8]:

```
# Bilevel training (unrolled) with balanced metric-weighted loss
```

```
def train_taskmet_unrolled_classification(
    tr_loader, val_loader, classifier, metric_net, label_encoder, fixed_class_weights,
    device="cpu", inner_steps=5, inner_lr=1e-2, outer_lr=1e-4, epochs=5, sync_alpha=0.1
):
    classifier.to(device)
    metric_net.to(device)
    fixed_class_weights = fixed_class_weights.to(device)
    outer_opt = torch.optim.Adam(metric_net.parameters(), lr=outer_lr)

    for ep in range(1, epochs + 1):
        classifier.train()
        metric_net.train()
        running_loss = 0.0
        nb = 0
        for X_batch, y_batch in tr_loader:
            X_batch = X_batch.to(device)
            y_batch = y_batch.to(device)

            # clone classifier for inner updates
            inner_model = copy.deepcopy(classifier).to(device)

            # unrolled inner updates
            for k in range(inner_steps):
                logits = inner_model(X_batch)                          # (B, C)
                metric_w = metric_net(X_batch)                         # (B, C)
                inner_loss = metricized_crossentropy_balanced(logits, y_batch, metric_w,
                grads = torch.autograd.grad(inner_loss, inner_model.parameters(), create_
                with torch.no_grad():
                    for p, g in zip(inner_model.parameters(), grads):
                        if g is None:
                            continue
                        p -= inner_lr * g

            # outer loss: based on final inner_model predictions and metric_net
            logits_final = inner_model(X_batch)
            metric_w_final = metric_net(X_batch)
            outer_loss = metricized_crossentropy_balanced(logits_final, y_batch, metric_

            outer_opt.zero_grad()
            outer_loss.backward()
            outer_opt.step()

            # soft-sync classifier toward inner_model
            with torch.no_grad():
                for p_main, p_inner in zip(classifier.parameters(), inner_model.paramete
                    p_main.data.mul_(1.0 - sync_alpha).add_(sync_alpha * p_inner.data)

            running_loss += outer_loss.item() * X_batch.shape[0]
            nb += X_batch.shape[0]

        avg_loss = running_loss / max(1, nb)
        val_acc = evaluate_classifier(classifier, val_loader, device)
        print(f"[Epoch {ep}] train_task_loss={avg_loss:.4f} | val_acc={val_acc:.4f}")

    return classifier, metric_net


# Baseline classifier training (cross-entropy)

def train_baseline_classifier(model, tr_loader, val_loader, device="cpu", epochs=5, lr=1
    model.to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)
    if class_weights is not None:
        criterion = nn.CrossEntropyLoss(weight=class_weights.to(device))
    else:
        criterion = nn.CrossEntropyLoss()
    for ep in range(1, epochs + 1):
        model.train()
```

```
        total_loss = 0.0
        n = 0
        for X, y in tr_loader:
            X = X.to(device); y = y.to(device)
            optimizer.zero_grad()
            logits = model(X)
            loss = criterion(logits, y)
            loss.backward()
            optimizer.step()
            total_loss += loss.item() * X.shape[0]
            n += X.shape[0]
        avg_loss = total_loss / max(1, n)
        val_acc = evaluate_classifier(model, val_loader, device)
        print(f"[Baseline E{ep}] train_loss={avg_loss:.4f} val_acc={val_acc:.4f}")
    return model
```

In [ ]:

```
import math
# Main Function
def main():
    parser = argparse.ArgumentParser()
    args = parser.parse_args([])
    args.data_dir = './da5401-2025-data-challenge'
    args.device = 'cuda'
    args.epochs = 100
    args.batch_size = 16
    args.out_dir = './output_taskmet'
    args.seed = 42
    args.embed_model = "google/embeddinggemma-300m"
    args.mode = 'taskmet'
    args.inner_steps = 5
    args.inner_lr = 1e-2
    args.outer_lr = 1e-4

    #random.seed(args.seed); np.random.seed(args.seed); torch.manual_seed(args.seed)

    os.makedirs(args.out_dir, exist_ok=True)
    cache_dir = os.path.join(args.out_dir, "cache")
    tr_loader, val_loader, test_loader, input_dim, label_encoder, y_train_cls_all = prep
        args.data_dir, args.embed_model, args.device, cache_dir, val_frac=0.1, batch_siz
    )

    num_classes = len(label_encoder.classes_)
    print("num_classes:", num_classes, "feature_dim:", input_dim)

    fixed_class_weights = compute_class_balanced_weights(y_train_cls_all, num_classes=nu
    print("Fixed class-balanced weights (normalized mean=1):", fixed_class_weights.numpy

    if args.mode == "baseline":
        print("Training baseline classifier (class-weighted)...")
        model = MLPClassifier(input_dim, hidden=[1024,512], num_classes=num_classes, dro
        # optional: use fixed_class_weights in CrossEntropyLoss to help imbalance
        model = train_baseline_classifier(model, tr_loader, val_loader, device=args.devi
        torch.save(model.state_dict(), os.path.join(args.out_dir, "baseline_classifier.p
        save_submission(model, test_loader, label_encoder, os.path.join(args.out_dir, "s

    else:
        print("Training TaskMet (classification, unrolled + balanced)...")
        classifier = MLPClassifier(input_dim, hidden=[1024,512], num_classes=num_classes
        metric_net = MetricNet(input_dim, num_classes=num_classes, hidden=[256]).to(args
        classifier, metric_net = train_taskmet_unrolled_classification(
            tr_loader, val_loader, classifier, metric_net, label_encoder, fixed_class_we
            device=args.device, inner_steps=args.inner_steps, inner_lr=args.inner_lr, ou
        )
        torch.save(classifier.state_dict(), os.path.join(args.out_dir, "taskmet_classifi
        torch.save(metric_net.state_dict(), os.path.join(args.out_dir, "taskmet_metricne
        save_submission(classifier, test_loader, label_encoder, os.path.join(args.out_di
```

```
if __name__ == "__main__":
    from huggingface_hub import login
    login("User_access_token")
    main()
```

```
Loading cached text embeddings: ./output_taskmet/cache/all_text_emb.npy
Unique training scores: [0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 9.5, 10.0]
num_classes: 12 feature_dim: 1536
Fixed class-balanced weights (normalized mean=1): [3.1281176e-01 6.7657441e-01 8.1168640e
-01 5.8006591e-01 1.3521345e+00
 4.0543756e+00 9.1092102e-02 4.3688588e-02 1.6685518e-02 2.5652135e-03
 4.0543756e+00 3.9453120e-03]
Training TaskMet (classification, unrolled + balanced)...
[Epoch 1] train_task_loss=0.0180 | val_acc=0.0200
[Epoch 2] train_task_loss=0.0122 | val_acc=0.0200
[Epoch 3] train_task_loss=0.0078 | val_acc=0.0200
[Epoch 4] train_task_loss=0.0052 | val_acc=0.0200
[Epoch 5] train_task_loss=0.0035 | val_acc=0.0200
[Epoch 6] train_task_loss=0.0025 | val_acc=0.0200
[Epoch 7] train_task_loss=0.0018 | val_acc=0.0200
[Epoch 8] train_task_loss=0.0013 | val_acc=0.0200
[Epoch 9] train_task_loss=0.0010 | val_acc=0.0200
[Epoch 10] train_task_loss=0.0007 | val_acc=0.0200
[Epoch 11] train_task_loss=0.0006 | val_acc=0.0200
[Epoch 12] train_task_loss=0.0004 | val_acc=0.0200
[Epoch 13] train_task_loss=0.0003 | val_acc=0.0200
[Epoch 14] train_task_loss=0.0003 | val_acc=0.0200
[Epoch 15] train_task_loss=0.0002 | val_acc=0.0200
[Epoch 16] train_task_loss=0.0002 | val_acc=0.0200
[Epoch 17] train_task_loss=0.0001 | val_acc=0.0200
[Epoch 18] train_task_loss=0.0001 | val_acc=0.0200
[Epoch 19] train_task_loss=0.0001 | val_acc=0.0200
[Epoch 20] train_task_loss=0.0001 | val_acc=0.0200
[Epoch 21] train_task_loss=0.0001 | val_acc=0.0200
[Epoch 22] train_task_loss=0.0001 | val_acc=0.0200
[Epoch 23] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 24] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 25] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 26] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 27] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 28] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 29] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 30] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 31] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 32] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 33] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 34] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 35] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 36] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 37] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 38] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 39] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 40] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 41] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 42] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 43] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 44] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 45] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 46] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 47] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 48] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 49] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 50] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 51] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 52] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 53] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 54] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 55] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 56] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 57] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 58] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 59] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 60] train_task_loss=0.0000 | val_acc=0.0200
```

```
[Epoch 61] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 62] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 63] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 64] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 65] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 66] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 67] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 68] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 69] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 70] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 71] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 72] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 73] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 74] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 75] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 76] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 77] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 78] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 79] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 80] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 81] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 82] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 83] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 84] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 85] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 86] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 87] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 88] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 89] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 90] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 91] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 92] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 93] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 94] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 95] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 96] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 97] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 98] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 99] train_task_loss=0.0000 | val_acc=0.0200
[Epoch 100] train_task_loss=0.0000 | val_acc=0.0200
Saved submission to: ./output_taskmet/submission_taskmet.csv
```

In [ ]:

```python
import math
# Main Function
def main():
    parser = argparse.ArgumentParser()
    args = parser.parse_args([])
    args.data_dir = './da5401-2025-data-challenge'
    args.device = 'cuda'
    args.epochs = 20
    args.batch_size = 16
    args.out_dir = './output_taskmet'
    args.seed = 42
    args.embed_model = "google/embeddinggemma-300m"
    args.mode = 'baseline'
    args.inner_steps = 5
    args.inner_lr = 1e-2
    args.outer_lr = 1e-4

    #random.seed(args.seed); np.random.seed(args.seed); torch.manual_seed(args.seed)

    os.makedirs(args.out_dir, exist_ok=True)
    cache_dir = os.path.join(args.out_dir, "cache")
    tr_loader, val_loader, test_loader, input_dim, label_encoder, y_train_cls_all = prep
        args.data_dir, args.embed_model, args.device, cache_dir, val_frac=0.1, batch_siz
    )

    num_classes = len(label_encoder.classes_)
```

```
    print("num_classes:", num_classes, "feature_dim:", input_dim)

    fixed_class_weights = compute_class_balanced_weights(y_train_cls_all, num_classes=nu
    print("Fixed class-balanced weights (normalized mean=1):", fixed_class_weights.numpy

    if args.mode == "baseline":
        print("Training baseline classifier (class-weighted)...")
        model = MLPClassifier(input_dim, hidden=[1024,512], num_classes=num_classes, dro
        # optional: use fixed_class_weights in CrossEntropyLoss to help imbalance
        model = train_baseline_classifier(model, tr_loader, val_loader, device=args.devi
        torch.save(model.state_dict(), os.path.join(args.out_dir, "baseline_classifier.p
        save_submission(model, test_loader, label_encoder, os.path.join(args.out_dir, "s

    else:
        print("Training TaskMet (classification, unrolled + balanced)...")
        classifier = MLPClassifier(input_dim, hidden=[1024,512], num_classes=num_classes
        metric_net = MetricNet(input_dim, num_classes=num_classes, hidden=[256]).to(args
        classifier, metric_net = train_taskmet_unrolled_classification(
            tr_loader, val_loader, classifier, metric_net, label_encoder, fixed_class_we
            device=args.device, inner_steps=args.inner_steps, inner_lr=args.inner_lr, ou
        )
        torch.save(classifier.state_dict(), os.path.join(args.out_dir, "taskmet_classifi
        torch.save(metric_net.state_dict(), os.path.join(args.out_dir, "taskmet_metricne
        save_submission(classifier, test_loader, label_encoder, os.path.join(args.out_di

if __name__ == "__main__":
    from huggingface_hub import login
    login("User_access_token")
    main()
```

```
Loading cached text embeddings: ./output_taskmet/cache/all_text_emb.npy
Unique training scores: [0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 9.5, 10.0]
num_classes: 12 feature_dim: 1536
Fixed class-balanced weights (normalized mean=1): [3.1281176e-01 6.7657441e-01 8.1168640e
-01 5.8006591e-01 1.3521345e+00
 4.0543756e+00 9.1092102e-02 4.3688588e-02 1.6685518e-02 2.5652135e-03
 4.0543756e+00 3.9453120e-03]
Training baseline classifier (class-weighted)...
[Baseline E1] train_loss=1.9599 val_acc=0.6380
[Baseline E2] train_loss=1.7599 val_acc=0.6520
[Baseline E3] train_loss=1.6515 val_acc=0.6500
[Baseline E4] train_loss=1.5920 val_acc=0.6260
[Baseline E5] train_loss=1.5413 val_acc=0.6080
[Baseline E6] train_loss=1.4878 val_acc=0.5760
[Baseline E7] train_loss=1.4286 val_acc=0.6300
[Baseline E8] train_loss=1.3729 val_acc=0.5760
[Baseline E9] train_loss=1.3508 val_acc=0.6080
[Baseline E10] train_loss=1.2745 val_acc=0.5520
[Baseline E11] train_loss=1.2359 val_acc=0.5620
[Baseline E12] train_loss=1.1828 val_acc=0.5620
[Baseline E13] train_loss=1.1353 val_acc=0.5980
[Baseline E14] train_loss=1.0778 val_acc=0.5240
[Baseline E15] train_loss=1.0515 val_acc=0.4960
[Baseline E16] train_loss=1.0155 val_acc=0.5240
[Baseline E17] train_loss=0.9735 val_acc=0.4900
[Baseline E18] train_loss=0.9089 val_acc=0.5480
[Baseline E19] train_loss=0.8806 val_acc=0.5400
[Baseline E20] train_loss=0.8534 val_acc=0.5160
Saved submission to: ./output_taskmet/submission_baseline.csv
```

In [ ]:

```
import math
# Main Function
def main():
    parser = argparse.ArgumentParser()
    args = parser.parse_args([])
    args.data_dir = './da5401-2025-data-challenge'
    args.device = 'cuda'
    args.epochs = 20
```

```
    args.batch_size = 16
    args.out_dir = './output_taskmet'
    args.seed = 42
    args.embed_model = "google/embeddinggemma-300m"
    args.mode = 'taskmet'
    args.inner_steps = 5
    args.inner_lr = 1e-2
    args.outer_lr = 1e-4

    #random.seed(args.seed); np.random.seed(args.seed); torch.manual_seed(args.seed)

    os.makedirs(args.out_dir, exist_ok=True)
    cache_dir = os.path.join(args.out_dir, "cache")
    tr_loader, val_loader, test_loader, input_dim, label_encoder, y_train_cls_all = prep
        args.data_dir, args.embed_model, args.device, cache_dir, val_frac=0.1, batch_siz
    )

    num_classes = len(label_encoder.classes_)
    print("num_classes:", num_classes, "feature_dim:", input_dim)

    fixed_class_weights = compute_class_balanced_weights(y_train_cls_all, num_classes=nu
    print("Fixed class-balanced weights (normalized mean=1):", fixed_class_weights.numpy

    if args.mode == "baseline":
        print("Training baseline classifier (class-non weighted)...")
        model = MLPClassifier(input_dim, hidden=[1024,512], num_classes=num_classes, dro
        # optional: use fixed_class_weights in CrossEntropyLoss to help imbalance
        model = train_baseline_classifier(model, tr_loader, val_loader, device=args.devi
        torch.save(model.state_dict(), os.path.join(args.out_dir, "baseline_classifier.p
        save_submission(model, test_loader, label_encoder, os.path.join(args.out_dir, "s

    else:
        print("Training TaskMet (classification, unrolled + balanced)...")
        classifier = MLPClassifier(input_dim, hidden=[1024,512], num_classes=num_classes
        metric_net = MetricNet(input_dim, num_classes=num_classes, hidden=[256]).to(args
        classifier, metric_net = train_taskmet_unrolled_classification(
            tr_loader, val_loader, classifier, metric_net, label_encoder, fixed_class_we
            device=args.device, inner_steps=args.inner_steps, inner_lr=args.inner_lr, ou
        )
        torch.save(classifier.state_dict(), os.path.join(args.out_dir, "taskmet_classifi
        torch.save(metric_net.state_dict(), os.path.join(args.out_dir, "taskmet_metricne
        save_submission(classifier, test_loader, label_encoder, os.path.join(args.out_di

if __name__ == "__main__":
    from huggingface_hub import login
    login("User_access_token")
    main()
```

```
Loading cached text embeddings: ./output_taskmet/cache/all_text_emb.npy
Unique training scores: [0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 9.5, 10.0]
num_classes: 12 feature_dim: 1536
Fixed class-balanced weights (normalized mean=1): [3.1281176e-01 6.7657441e-01 8.1168640e
-01 5.8006591e-01 1.3521345e+00
 4.0543756e+00 9.1092102e-02 4.3688588e-02 1.6685518e-02 2.5652135e-03
 4.0543756e+00 3.9453120e-03]
Training TaskMet (classification, unrolled + balanced)...
[Epoch 1] train_task_loss=0.0176 | val_acc=0.1480
[Epoch 2] train_task_loss=0.0117 | val_acc=0.1460
[Epoch 3] train_task_loss=0.0071 | val_acc=0.1340
[Epoch 4] train_task_loss=0.0045 | val_acc=0.1320
[Epoch 5] train_task_loss=0.0029 | val_acc=0.1260
[Epoch 6] train_task_loss=0.0020 | val_acc=0.1300
[Epoch 7] train_task_loss=0.0014 | val_acc=0.1280
[Epoch 8] train_task_loss=0.0010 | val_acc=0.1280
[Epoch 9] train_task_loss=0.0007 | val_acc=0.1280
[Epoch 10] train_task_loss=0.0005 | val_acc=0.1280
[Epoch 11] train_task_loss=0.0004 | val_acc=0.1280
[Epoch 12] train_task_loss=0.0003 | val_acc=0.1280
[Epoch 13] train_task_loss=0.0003 | val_acc=0.1280
[Epoch 14] train_task_loss=0.0002 | val_acc=0.1280
[Epoch 15] train_task_loss=0.0002 | val_acc=0.1280
[Epoch 16] train_task_loss=0.0001 | val_acc=0.1280
[Epoch 17] train_task_loss=0.0001 | val_acc=0.1280
[Epoch 18] train_task_loss=0.0001 | val_acc=0.1280
[Epoch 19] train_task_loss=0.0001 | val_acc=0.1280
[Epoch 20] train_task_loss=0.0001 | val_acc=0.1280
Saved submission to: ./output_taskmet/submission_taskmet.csv
```

In [ ]:

```python
# Compare Submission
from sklearn.metrics import confusion_matrix,classification_report
sample_path = '/kaggle/input/data-5401/sample_submission.csv'
output_path = '/kaggle/working/output/submission_taskmet.csv'
act_labels = pd.read_csv(sample_path)['score']
pred_labels = pd.read_csv(output_path)['score']
mse = np.mean((pred_labels - act_labels)**2)
rmse = np.sqrt(mse)
#confusion_matrix(act_labels, pred_labels)
print(classification_report(act_labels, pred_labels))
print("rmse:",rmse)
```