Analysing MQM error profiles and metric correlations in MT systems

Multilingual NLP – Lab 3

Philippos Triantafyllou

Table of contents

1 Data loading

```python
import pandas as pd

pd.set_option("display.max_colwidth", 20)
```

```python
# Load the TSV file
df = pd.read_csv("data/mqm_2022_ende.tsv", sep="\t")

# # Select required columns
cols = ["system_id", "doc_segment_id", "category", "severity", "source", "candidate", "reference"]
df = df[cols]
df = df.rename(columns={'system_id': 'system', 'candidate': 'target', 'doc_segment_id': 'segment'})

display(df)
print(df.shape)
```

	system	segment	category	severity	source	target
0	JDExploreAcademy	1	style/awkward	minor	Avalanche at Was…	Lawine in Ski
1	JDExploreAcademy	1	style/awkward	minor	Avalanche at Was…	Lawine in Ski
2	JDExploreAcademy	1	no-error	no-error	Avalanche at Was…	Lawine in Ski
3	JDExploreAcademy	1	accuracy/mistran…	minor	Avalanche at Was…	Lawine in Ski
4	Lan-Bridge	1	style/awkward	minor	Avalanche at Was…	Lawine im Ski
…	…	…	…	…	…	…
78554	comet_bestmbr	10	no-error	no-error	I shall never fo…	Ich werde dies
78555	refB	10	fluency/punctuation	minor	I shall never fo…	<v>Ich</v>
78556	refB	10	fluency/punctuation	minor	I shall never fo…	Ich werde dies
78557	refB	10	no-error	no-error	I shall never fo…	Ich werde dies
78558	refB	10	no-error	no-error	I shall never fo…	Ich werde dies

```
(78559, 7)
```

Doing some cleaning.

```python
# Drop rows with missing annotations in key fields
df = df.dropna(subset=["system", "segment", "category", "severity", "source", "target",
↪  "reference"]).reset_index(drop=True)
display(df.shape)
```

```
(78559, 7)
```

```python
# View the unique category and severity labels
print("Unique Categories:", df["category"].unique())
print("Unique Severities:", df["severity"].unique())
```

```
Unique Categories: ['style/awkward' 'no-error' 'accuracy/mistranslation'
 'accuracy/untranslated' 'fluency/grammar' 'accuracy/addition'
 'accuracy/omission' 'terminology/inappropriate' 'locale_convention/time'
 'fluency/register' 'terminology/inconsistent' 'fluency/spelling'
 'fluency/punctuation' 'fluency/display' 'fluency/inconsistency'
 'locale_convention/currency' 'source_error' 'other' 'non_translation'
 'locale_convention/date' 'locale_convention/name']
Unique Severities: ['minor' 'no-error' 'major']
```

```
# Build a major category variable
df["major_category"] = df["category"].apply(lambda x: x.split("/")[0].split(".")[0].strip() if pd.notnull(x)
↪  else x)
df["major_category"].unique()
```

```
array(['style', 'no-error', 'accuracy', 'fluency', 'terminology',
       'locale_convention', 'source_error', 'other', 'non_translation'],
      dtype=object)
```

The data is now ready to work with.

2 Building and visualizing error profiles

First we build for each system its error profile as a vector of error counts per major category.

```
df_err = df[df["major_category"] != "no-error"].copy()
```

```
# Count errors per system and category
error_counts = (df_err.groupby(["system", "major_category"]).size().unstack(fill_value=0))
```

```
display(error_counts)
```

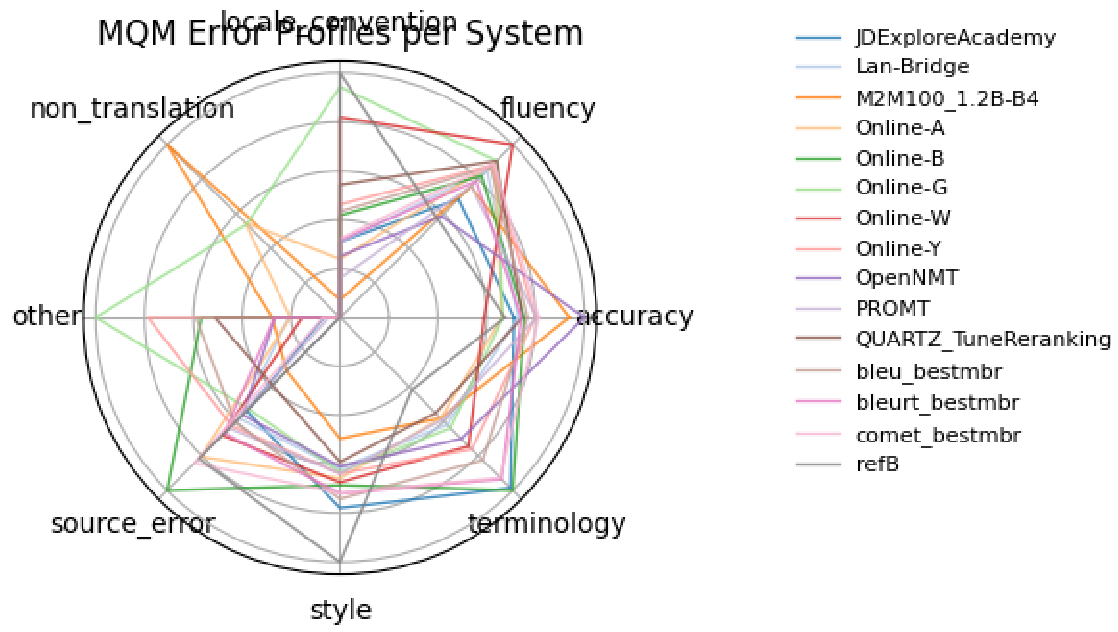
major_category system	accuracy	fluency	locale_convention	non_translation	other	source_error
JDExploreAcademy	957	874	4	0	1	14
Lan-Bridge	1233	1362	5	0	4	19
M2M100_1.2B-B4	2681	2054	2	3	9	17
Online-A	1151	1468	4	1	4	27
Online-B	940	967	5	0	8	24
Online-G	1177	1516	16	1	20	18
Online-W	673	1082	9	0	2	15
Online-Y	1256	1312	7	0	14	19
OpenNMT	1657	921	4	0	5	18
PROMT	1564	1399	3	0	1	24
QUARTZ_TuneReranking	1287	1482	9	0	10	13
bleu_bestmbr	920	988	5	0	8	14
bleurt_bestmbr	976	986	4	0	4	17
comet_bestmbr	1051	1092	4	0	1	21
refB	772	614	11	0	0	18

```
# count target-side tokens
df_err["target_tokens"] = df_err["target"].str.split().str.len()

tokens_per_system = (df_err.groupby("system")["target_tokens"].sum())

error_freq_per_token = error_counts.div(tokens_per_system, axis=0)
```

```
display(error_freq_per_token)
```

major_category system	accuracy	fluency	locale_convention	non_translation	other	source_e
JDExploreAcademy	0.017037	0.015560	0.000071	0.000000	0.000018	0.000249
Lan-Bridge	0.017461	0.019287	0.000071	0.000000	0.000057	0.000269
M2M100_1.2B-B4	0.022451	0.017200	0.000017	0.000025	0.000075	0.000142
Online-A	0.015857	0.020225	0.000055	0.000014	0.000055	0.000372
Online-B	0.018068	0.018587	0.000096	0.000000	0.000154	0.000461
Online-G	0.015959	0.020555	0.000217	0.000014	0.000271	0.000244
Online-W	0.014108	0.022682	0.000189	0.000000	0.000042	0.000314
Online-Y	0.019116	0.019969	0.000107	0.000000	0.000213	0.000289
OpenNMT	0.023920	0.013295	0.000058	0.000000	0.000072	0.000260
PROMT	0.019253	0.017222	0.000037	0.000000	0.000012	0.000295
QUARTZ_TuneReranking	0.017857	0.020562	0.000125	0.000000	0.000139	0.000180
bleu_bestmbr	0.018387	0.019746	0.000100	0.000000	0.000160	0.000280
bleurt_bestmbr	0.017721	0.017902	0.000073	0.000000	0.000073	0.000309
comet_bestmbr	0.019453	0.020212	0.000074	0.000000	0.000019	0.000389
refB	0.016153	0.012847	0.000230	0.000000	0.000000	0.000377

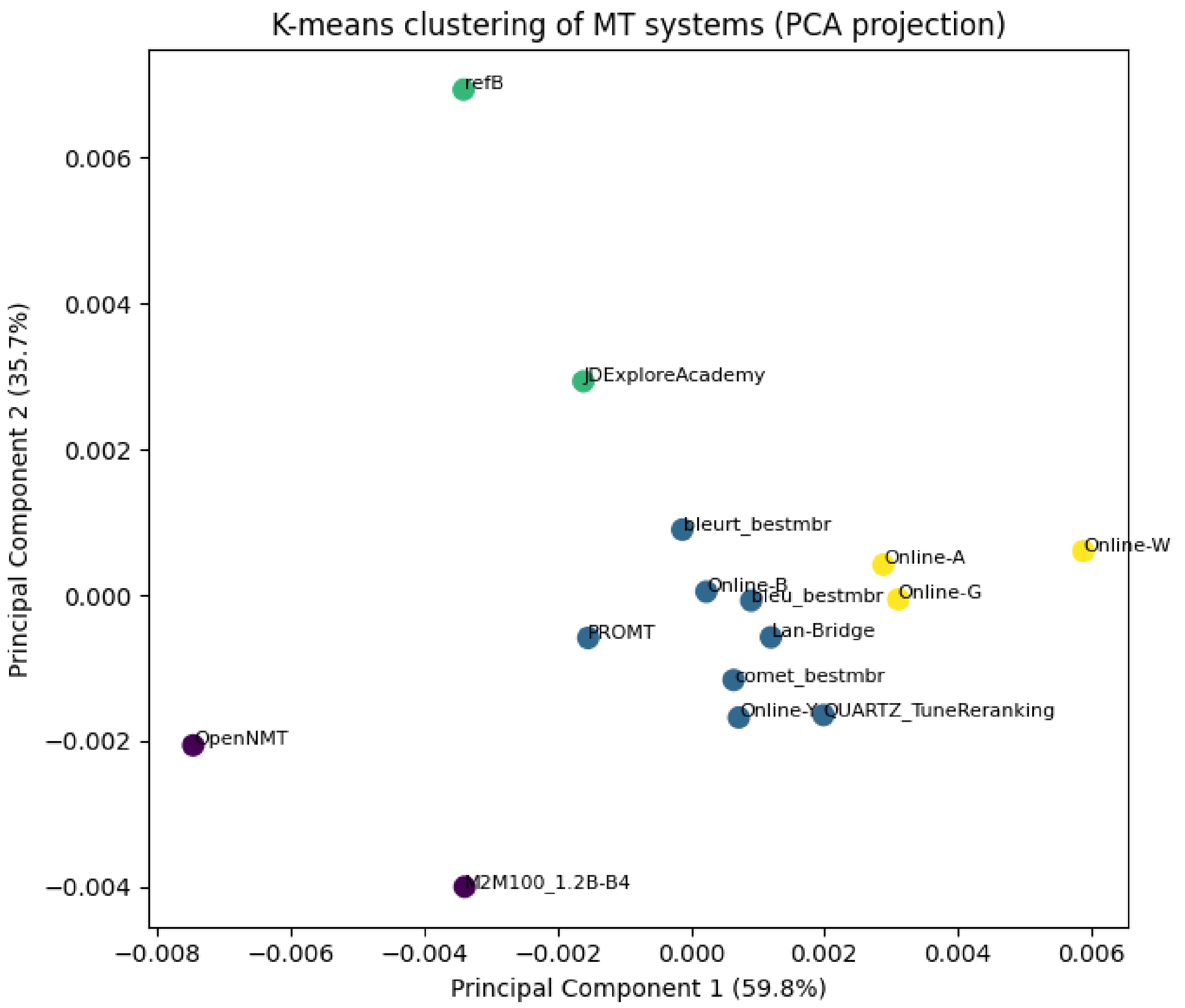
Then we visualize using a radar plot.



3 Clustering systems by error profiles

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

error_frequency_dataframe = error_freq_per_token
system_names = error_frequency_dataframe.index
feature_matrix = error_frequency_dataframe.values

number_of_clusters = 4
kmeans_model = KMeans(
    n_clusters=number_of_clusters,
```

```
    n_init=10,
    random_state=0,
)
cluster_labels = kmeans_model.fit_predict(feature_matrix)

pca_model = PCA(n_components=2)
pca_coordinates = pca_model.fit_transform(feature_matrix)

explained_variance = pca_model.explained_variance_ratio_

plt.figure(figsize=(7, 6))
plt.scatter(
    pca_coordinates[:, 0],
    pca_coordinates[:, 1],
    c=cluster_labels,
    s=70,
)

for i, system_name in enumerate(system_names):
    plt.text(
        pca_coordinates[i, 0],
        pca_coordinates[i, 1],
        system_name,
        fontsize=8,
    )

plt.xlabel(f"Principal Component 1 ({explained_variance[0]*100:.1f}%)")
plt.ylabel(f"Principal Component 2 ({explained_variance[1]*100:.1f}%)")
plt.title("K-means clustering of MT systems (PCA projection)")
plt.tight_layout()
plt.show()
```



4 MQM and Automatic Metrics

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sacrebleu
import torch

from scipy.stats import pearsonr, spearmanr
from comet import load_from_checkpoint
from comet.models import download_model
```

```
import pandas as pd

# Define the required columns for metric computation
required_columns = ["system", "segment", "source", "target", "reference"]

# Keep only the columns needed to compute segment/system metrics
metrics_input_dataframe = df[required_columns].copy()

# Drop rows missing any of the text fields needed for metrics
metrics_input_dataframe = metrics_input_dataframe.dropna(subset=["source", "target", "reference"])

# Build the segment-level table (one row per system-segment)
segment_level_metrics_table = metrics_input_dataframe.drop_duplicates(subset=["system", "segment"])

# Build the system-level table skeleton (one row per system)
system_level_metrics_table = (
    segment_level_metrics_table.groupby("system", as_index=False)
    .agg(num_segments=("segment", "nunique"))
)
```

Computing weighted MQM scores following Freitag et al.

```
# Severity weighting scheme
severity_weights = {
    "minor": 1,
    "major": 5,
}

# Map severities to weights
# "no-error" -> NaN (ignored automatically)
df["mqm_weight"] = df["severity"].map(severity_weights)
```

Segment level MQM scores.

```
mqm_segment_scores = df.groupby(["system",
 ↪  "segment"])["mqm_weight"].sum(min_count=1).reset_index(name="mqm_score")
segment_level_metrics_table = segment_level_metrics_table.merge(mqm_segment_scores, on=["system",
 ↪  "segment"], how="left")
```

System level MQM scores normalized by total tokens per system.

```
df["target_tokens"] = df["target"].str.split().str.len()
tokens_per_system = (df.groupby("system")["target_tokens"].sum())
mqm_system_scores = (df.groupby("system")["mqm_weight"].sum(min_count=1))
mqm_system_scores_norm = mqm_system_scores / tokens_per_system

system_level_metrics_table = system_level_metrics_table.merge(
    mqm_system_scores_norm.reset_index(name="mqm_score_per_token"),
    on="system",
    how="left",
)
```

Computing BLEU and chrF scores at the segment level.

```
bleu_scores = []
chrf_scores = []

for _, row in segment_level_metrics_table.iterrows():
    bleu = sacrebleu.sentence_bleu(row["target"], [row["reference"]]).score
    chrf = sacrebleu.sentence_chrf(row["target"], [row["reference"]]).score
    bleu_scores.append(bleu)
    chrf_scores.append(chrf)

# Add segment-level BLEU and chrF to the segment-level metrics table
segment_level_metrics_table["BLEU"] = bleu_scores
segment_level_metrics_table["chrF"] = chrf_scores
```

System level BLEU and chrF scores.

```
# Compute corpus-level BLEU and chrF for each system using the segment-level metrics table
system_metric_results = []

for system_name in segment_level_metrics_table["system"].unique():
    system_segments = segment_level_metrics_table[segment_level_metrics_table["system"] == system_name]

    # Collect hypotheses and references for the full system output
    hypotheses = system_segments["target"].astype(str).tolist()
    references = [system_segments["reference"].astype(str).tolist()]

    # Compute corpus-level BLEU (system-level)
    bleu_score = sacrebleu.corpus_bleu(hypotheses, references).score

    # Compute corpus-level chrF (system-level)
    chrf_score = sacrebleu.corpus_chrf(hypotheses, references).score

    system_metric_results.append({
        "system": system_name,
        "BLEU": bleu_score,
        "chrF": chrf_score,
    })

# Put BLEU/chrF results into a separate table (one row per system)
system_level_bleu_chrf_table = pd.DataFrame(system_metric_results)

# Merge BLEU/chrF into the existing system-level metrics table (do NOT overwrite it)
system_level_metrics_table = system_level_metrics_table.merge(system_level_bleu_chrf_table, on="system",
↪ how="left")
```

COMET score.

```
# Load the COMET-DA model (reference-based, suitable for MQM correlation)
model_path = download_model("Unbabel/wmt22-comet-da")
comet_model = load_from_checkpoint(model_path)

# Prepare COMET inputs from the segment-level metrics table (one row per system-segment)
comet_inputs = [
    {
        "src": row["source"],
        "mt": row["target"],
        "ref": row["reference"],
    }
    for _, row in segment_level_metrics_table.iterrows()
]

# Run COMET to obtain segment-level scores
comet_output = comet_model.predict(
    comet_inputs,
    batch_size=32,
```

```
    gpus=1 if torch.cuda.is_available() else 0,
    num_workers=1,
)

# Add segment-level COMET scores to the segment-level metrics table
segment_level_metrics_table["COMET"] = comet_output["scores"]

# Aggregate system-level COMET as the mean of segment-level scores
comet_system_scores = segment_level_metrics_table.groupby("system")["COMET"].mean()

# Merge system-level COMET into the existing system-level metrics table
system_level_metrics_table = system_level_metrics_table.merge(
    comet_system_scores.rename("COMET"),
    on="system",
    how="left",
)
```

```
display(segment_level_metrics_table)
display(system_level_metrics_table)
```

	system	segment	source	target	reference
0	JDExploreAcademy	1	Avalanche at Was...	Lawine in Skigeb...	Lawine in Ski-Re...
1	Lan-Bridge	1	Avalanche at Was...	Lawine im Skigeb...	Lawine in Ski-Re...
2	M2M100_1.2B-B4	1	Avalanche at Was...	<v>Avalanche</v>...	Lawine in Ski-Re...
3	Online-A	1	Avalanche at Was...	Lawine im Skigeb...	Lawine in Ski-Re...
4	Online-B	1	Avalanche at Was...	Lawine im Skigeb...	Lawine in Ski-Re...
...	...	...	...	...	...
145	QUARTZ_TuneReran...	10	”Skiers can trav...	”Im Staatswald k...	„Skifahrer könne...
146	bleu_bestmbr	10	”Skiers can trav...	„Skifahrer könne...	„Skifahrer könne...
147	bleurt_bestmbr	10	”Skiers can trav...	„Skifahrer könne...	„Skifahrer könne...
148	comet_bestmbr	10	”Skiers can trav...	„Skifahrer könne...	„Skifahrer könne...
149	refB	10	”Skiers can trav...	„Skifahrer könne...	„Skifahrer könne...

	system	num_segments	mqm_score_per_token	BLEU	chrF	COMET
0	JDExploreAcademy	10	0.054599	32.666188	61.775083	0.822923
1	Lan-Bridge	10	0.064303	24.708933	59.038817	0.761160
2	M2M100_1.2B-B4	10	0.091230	21.711807	55.987241	0.721010
3	Online-A	10	0.060145	27.270501	59.721487	0.786887
4	Online-B	10	0.054561	25.004807	59.665942	0.770541
5	Online-G	10	0.062263	26.326593	60.875855	0.758391
6	Online-W	10	0.044980	34.177264	62.279745	0.845575
7	Online-Y	10	0.062435	25.471897	59.843116	0.740286
8	OpenNMT	10	0.075304	27.706626	60.139061	0.754235
9	PROMT	10	0.070187	25.850586	61.062388	0.763773
10	QUARTZ_TuneReran...	10	0.065215	16.270910	51.311063	0.793082
11	bleu_bestmbr	10	0.054473	28.446967	62.207114	0.784587
12	bleurt_bestmbr	10	0.055316	24.415303	58.161545	0.818754
13	comet_bestmbr	10	0.058619	26.081795	59.319196	0.830675
14	refB	10	0.047307	25.122366	59.532564	0.811405

System level correlations.

```
from scipy.stats import pearsonr, spearmanr

# Collect system-level correlation results in a structured table
system_correlation_results = []

for metric_name in ["BLEU", "chrF", "COMET"]:
    pearson_r, _ = pearsonr(
        system_level_metrics_table["mqm_score_per_token"],
        system_level_metrics_table[metric_name],
    )
    spearman_r, _ = spearmanr(
        system_level_metrics_table["mqm_score_per_token"],
        system_level_metrics_table[metric_name],
    )

    system_correlation_results.append({
        "metric": metric_name,
        "pearson_r": pearson_r,
        "spearman_r": spearman_r,
    })

# Convert to DataFrame
system_correlation_table = pd.DataFrame(system_correlation_results)

system_correlation_table
```

	metric	pearson_r	spearman_r
0	BLEU	-0.474604	-0.417857
1	chrF	-0.420903	-0.392857
2	COMET	-0.778326	-0.707143

Segment level correlations.

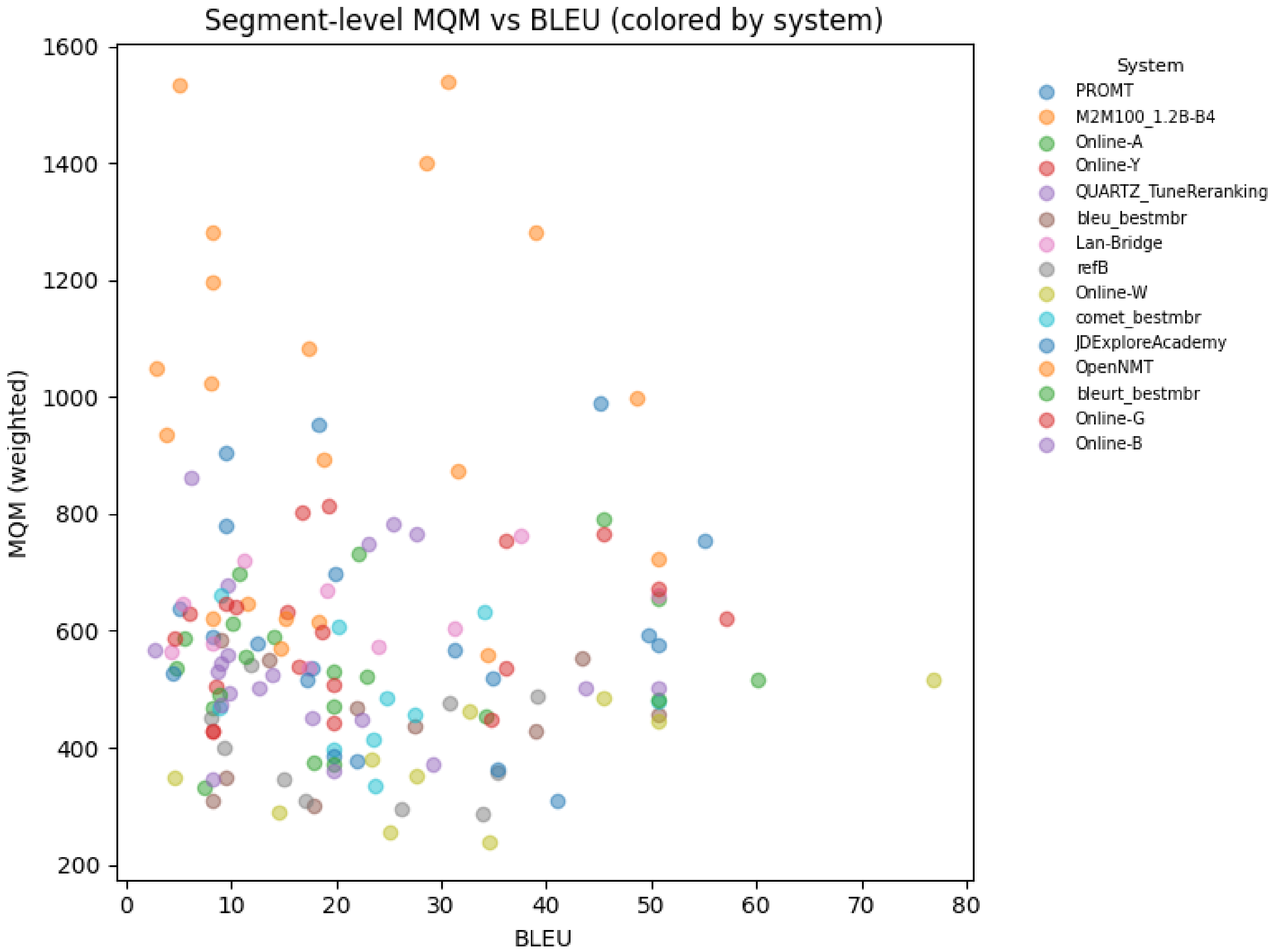
```
# Collect segment-level correlation results in a structured table
segment_correlation_results = []

for metric_name in ["BLEU", "chrF", "COMET"]:
    pearson_r, _ = pearsonr(
        segment_level_metrics_table["mqm_score"],
        segment_level_metrics_table[metric_name],
    )
    spearman_r, _ = spearmanr(
        segment_level_metrics_table["mqm_score"],
        segment_level_metrics_table[metric_name],
    )

    segment_correlation_results.append({
        "metric": metric_name,
        "pearson_r": pearson_r,
        "spearman_r": spearman_r,
    })

# Convert to DataFrame
segment_correlation_table = pd.DataFrame(segment_correlation_results)

segment_correlation_table
```

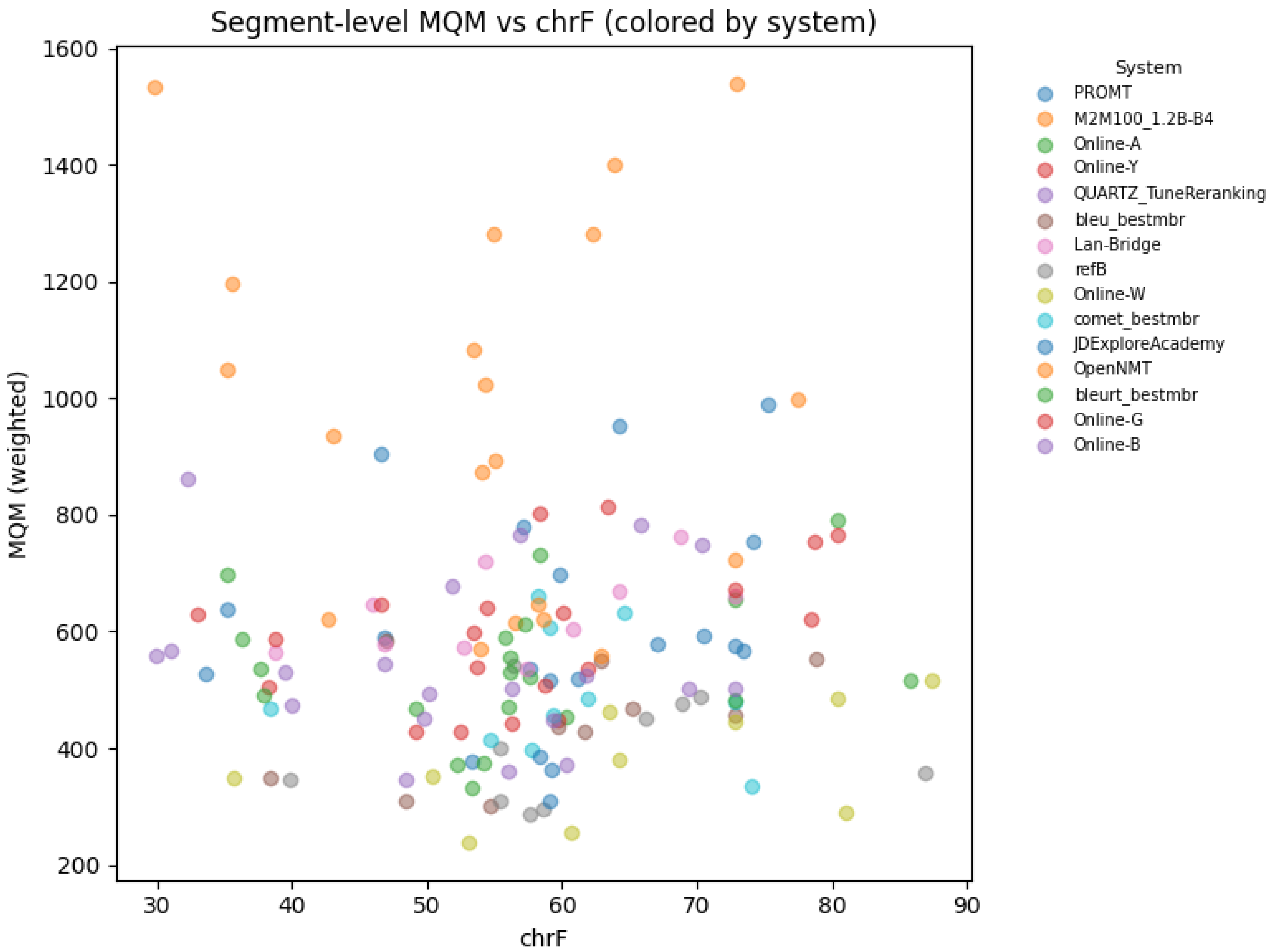
	metric	pearson_r	spearman_r
0	BLEU	-0.030795	-0.073729

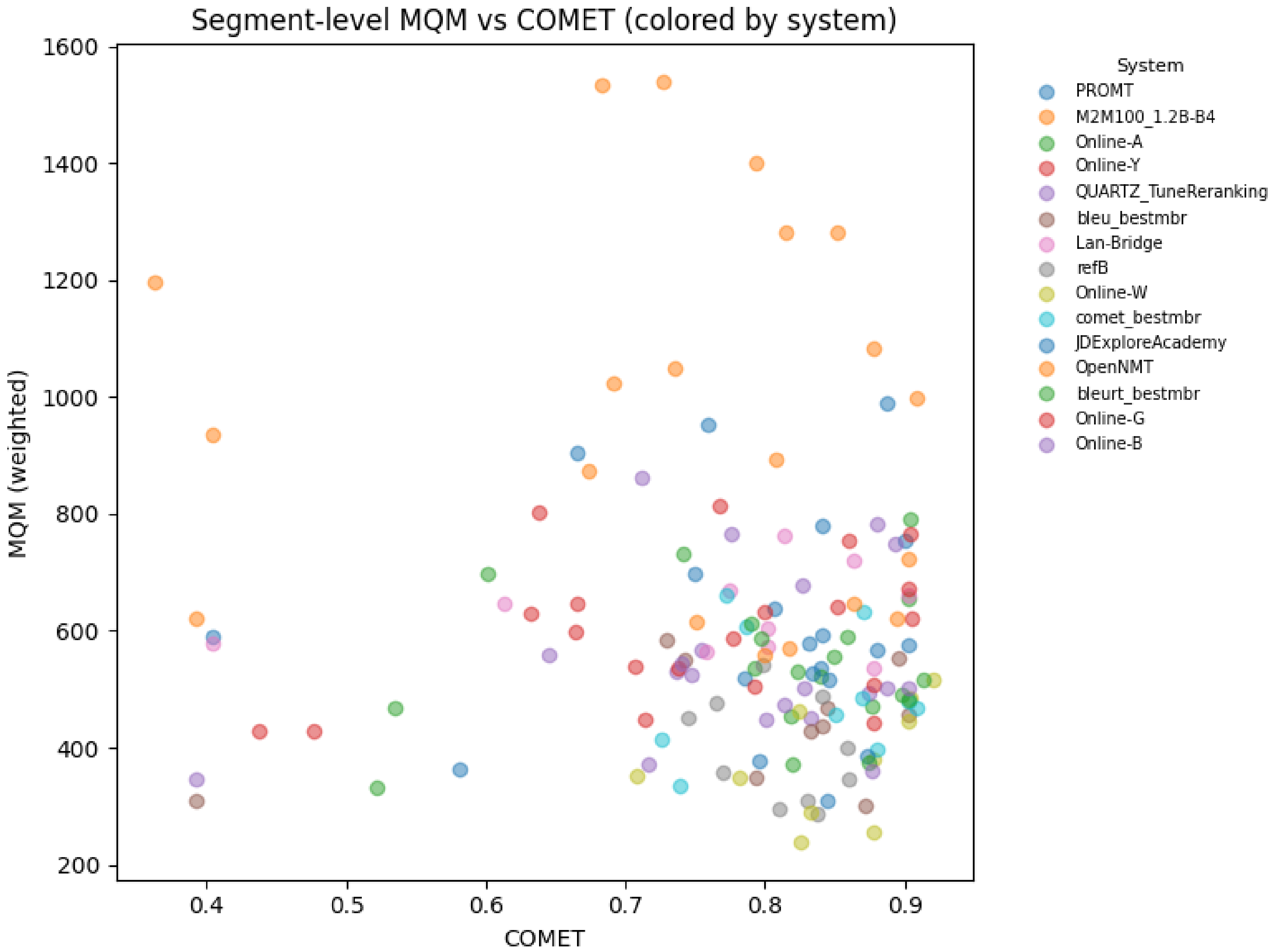
	metric	pearson_r	spearman_r
1	chrF	-0.024369	0.032225
2	COMET	-0.088716	-0.071358

Plotting segment level MQM vs automatic metrics.

```
import matplotlib.pyplot as plt

# Define how many points to plot (avoid overplotting)
sample_size = min(3000, len(segment_level_metrics_table))
sampled_segments = segment_level_metrics_table.sample(n=sample_size, random_state=0)

# Get unique systems
systems = sampled_segments["system"].unique()

# Plot segment-level MQM vs each automatic metric, colored by system with legend
for metric_name in ["BLEU", "chrF", "COMET"]:
    plt.figure(figsize=(8, 6))

    for system_name in systems:
        system_data = sampled_segments[sampled_segments["system"] == system_name]

        plt.scatter(
            system_data[metric_name],
            system_data["mqm_score"],
            alpha=0.5,
            label=system_name,
        )

    plt.xlabel(metric_name)
    plt.ylabel("MQM (weighted)")
    plt.title(f"Segment-level MQM vs {metric_name} (colored by system)")

    plt.legend(
        title="System",
        fontsize=7,
        title_fontsize=8,
        bbox_to_anchor=(1.05, 1),
        loc="upper left",
        frameon=False,
    )

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


Segment-level MQM vs BLEU (colored by system)
MQM (weighted)
BLEU
System
PROMT
M2M100_1.2B-B4
Online-A
Online-Y
QUARTZ_TuneReranking
bleu_bestmbr
Lan-Bridge
refB
Online-W
comet_bestmbr
JDExploreAcademy
OpenNMT
bleurt_bestmbr
Online-G
Online-B

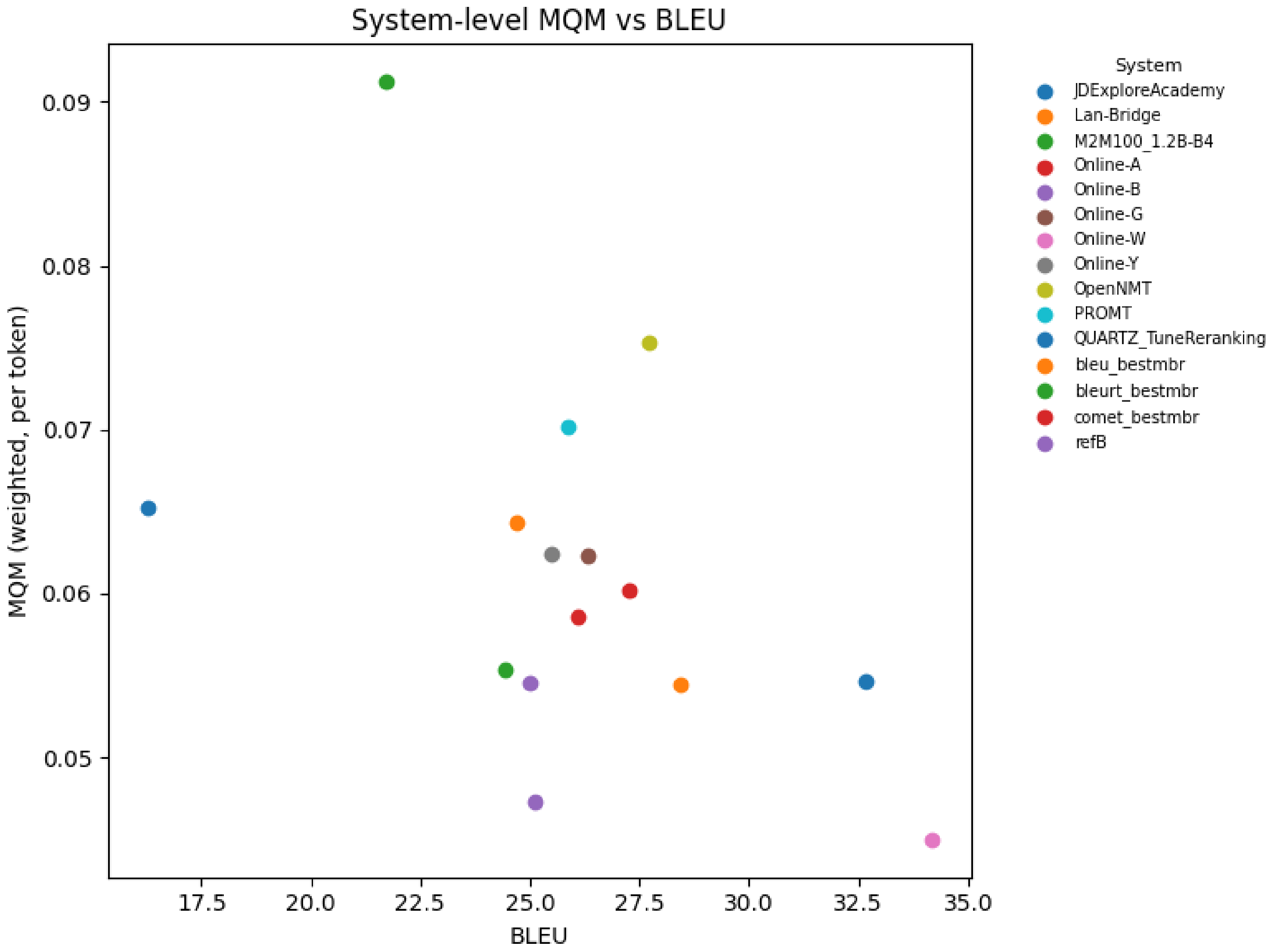

Segment-level MQM vs chrF (colored by system)
MQM (weighted)
chrF
System
PROMT
M2M100_1.2B-B4
Online-A
Online-Y
QUARTZ_TuneReranking
bleu_bestmbr
Lan-Bridge
refB
Online-W
comet_bestmbr
JDExploreAcademy
OpenNMT
bleurt_bestmbr
Online-G
Online-B

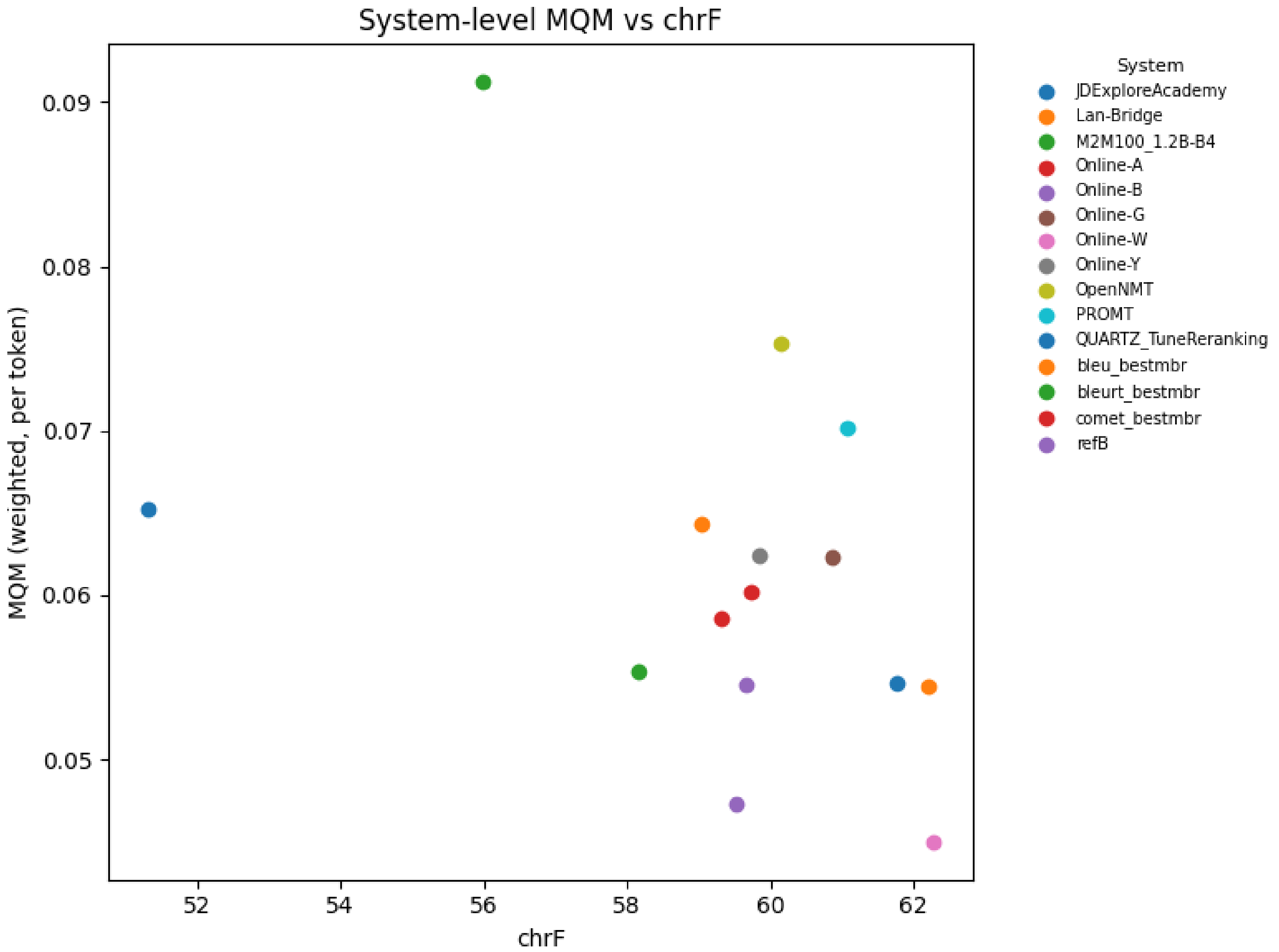


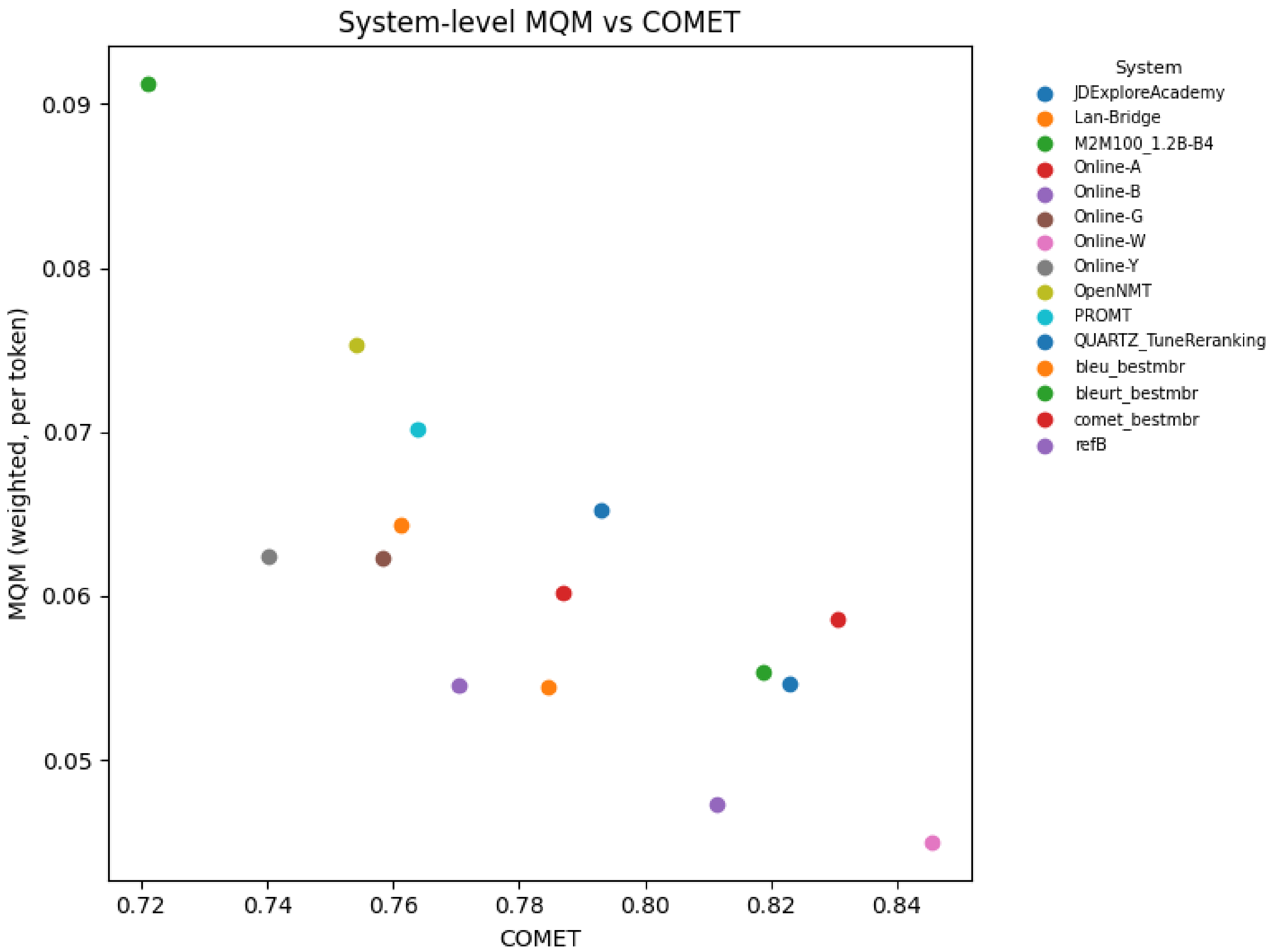
And at system level.

```
import matplotlib.pyplot as plt

# Get unique systems
systems = system_level_metrics_table["system"].unique()

# Plot system-level MQM vs each automatic metric, colored by system with legend
for metric_name in ["BLEU", "chrF", "COMET"]:
    plt.figure(figsize=(8, 6))

    for system_name in systems:
        system_data = system_level_metrics_table[system_level_metrics_table["system"] == system_name]

        plt.scatter(
            system_data[metric_name],
            system_data["mqm_score_per_token"],
            label=system_name,
        )

    plt.xlabel(metric_name)
    plt.ylabel("MQM (weighted, per token)")
    plt.title(f"System-level MQM vs {metric_name}")

    plt.legend(
        title="System",
        fontsize=7,
        title_fontsize=8,
        bbox_to_anchor=(1.05, 1),
        loc="upper left",
        frameon=False,
    )

    plt.tight_layout()
```

```
plt.show()
```


System-level MQM vs BLEU
MQM (weighted, per token)
BLEU
0.09
0.08
0.07
0.06
0.05
17.5
20.0
22.5
25.0
27.5
30.0
32.5
35.0
System
JDExploreAcademy
Lan-Bridge
M2M100_1.2B-B4
Online-A
Online-B
Online-G
Online-W
Online-Y
OpenNMT
PROMT
QUARTZ_TuneReranking
bleu_bestmbr
bleurt_bestmbr
comet_bestmbr
refB


System-level MQM vs chrF
MQM (weighted, per token)
0.09
0.08
0.07
0.06
0.05
52
54
56
58
60
62
chrF
System
JDExploreAcademy
Lan-Bridge
M2M100_1.2B-B4
Online-A
Online-B
Online-G
Online-W
Online-Y
OpenNMT
PROMT
QUARTZ_TuneReranking
bleu_bestmbr
bleurt_bestmbr
comet_bestmbr
refB


System-level MQM vs COMET
MQM (weighted, per token)
0.09
0.08
0.07
0.06
0.05
0.72
0.74
0.76
0.78
0.80
0.82
0.84
COMET
System
JDExploreAcademy
Lan-Bridge
M2M100_1.2B-B4
Online-A
Online-B
Online-G
Online-W
Online-Y
OpenNMT
PROMT
QUARTZ_TuneReranking
bleu_bestmbr
bleurt_bestmbr
comet_bestmbr
refB

5 Linguistic factors and correlation

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
```

Computing surface features:

- Character length of the source sentence
- Hedran's C.

```
# Character length of the source sentence
segment_level_metrics_table["char_length"] = segment_level_metrics_table["source"].astype(str).str.len()

# Tokenize source sentences (simple whitespace tokenization)
source_tokens = segment_level_metrics_table["source"].astype(str).str.split()

# Number of tokens per segment
segment_level_metrics_table["num_tokens"] = source_tokens.str.len()

# Number of unique tokens per segment
segment_level_metrics_table["num_types"] = source_tokens.apply(lambda toks: len(set(toks)) if toks else 0)

# Herdan's C: log(V) / log(N); undefined for N <= 1 → set to NaN
segment_level_metrics_table["herdan_c"] = np.where(
    segment_level_metrics_table["num_tokens"] > 1,
    np.log(segment_level_metrics_table["num_types"]) / np.log(segment_level_metrics_table["num_tokens"]),
    np.nan,
)
```

TF-IDF clustering of source sentences.

```
# Build TF-IDF vectors from source sentences
tfidf_vectorizer = TfidfVectorizer(
    lowercase=True,
    max_features=5000,
)

tfidf_matrix = tfidf_vectorizer.fit_transform(
    segment_level_metrics_table["source"].astype(str)
)

# Number of clusters (keep small and interpretable)
num_clusters = 5

kmeans_model = KMeans(
    n_clusters=num_clusters,
    n_init=10,
    random_state=0,
)

cluster_ids = kmeans_model.fit_predict(tfidf_matrix)

# Append cluster labels to the segment-level metrics table
segment_level_metrics_table["cluster_id"] = cluster_ids
```

Recomputing correlations per cluster.

```
import pandas as pd
from scipy.stats import pearsonr, spearmanr

# Collect TF-IDF cluster-level correlation results in a structured table
cluster_correlation_results = []
```

```python
for cluster_id in sorted(segment_level_metrics_table["cluster_id"].unique()):
    cluster_data = segment_level_metrics_table[
        segment_level_metrics_table["cluster_id"] == cluster_id
    ]

    for metric_name in ["BLEU", "chrF", "COMET"]:
        pearson_r, _ = pearsonr(
            cluster_data["mqm_score"],
            cluster_data[metric_name],
        )
        spearman_r, _ = spearmanr(
            cluster_data["mqm_score"],
            cluster_data[metric_name],
        )

        cluster_correlation_results.append({
            "cluster_id": cluster_id,
            "metric": metric_name,
            "num_segments": len(cluster_data),
            "pearson_r": pearson_r,
            "spearman_r": spearman_r,
        })

# Convert to DataFrame
tfidf_cluster_correlation_table = pd.DataFrame(cluster_correlation_results)

tfidf_cluster_correlation_table
```

	cluster_id	metric	num_segments	pearson_r	spearman_r
0	0	BLEU	45	-0.332387	-0.480482
1	0	chrF	45	-0.306500	-0.222808
2	0	COMET	45	-0.387375	-0.491600
3	1	BLEU	30	-0.105968	-0.140210
4	1	chrF	30	-0.197769	-0.134996
5	1	COMET	30	-0.177681	-0.120941
6	2	BLEU	15	-0.310248	-0.223870
7	2	chrF	15	-0.535192	-0.554563
8	2	COMET	15	-0.465373	-0.500895
9	3	BLEU	30	0.124528	0.132888
10	3	chrF	30	0.344018	0.335856
11	3	COMET	30	-0.057990	0.091445
12	4	BLEU	30	-0.350920	-0.477662
13	4	chrF	30	-0.264445	-0.367824
14	4	COMET	30	-0.292790	-0.235648

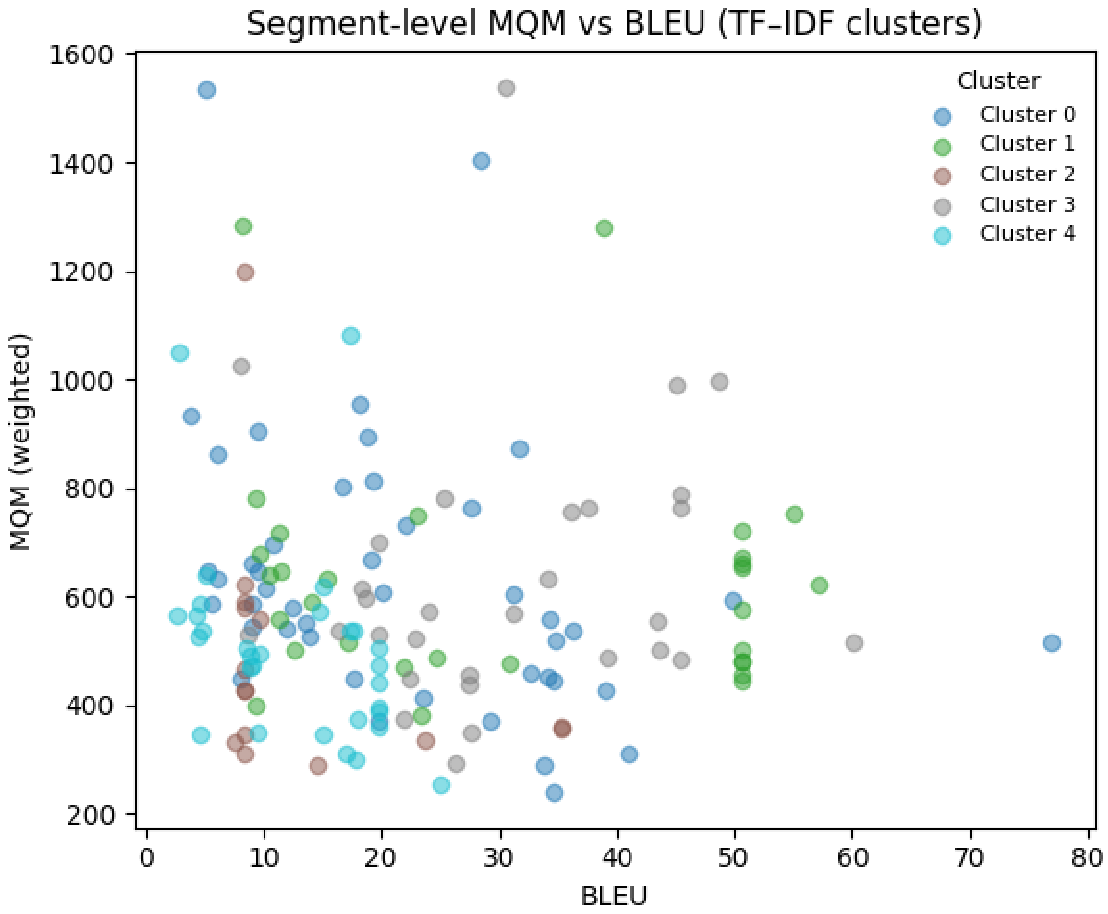
```python
import pandas as pd

# Compute descriptive statistics per TF-IDF cluster
cluster_statistics = (
    segment_level_metrics_table
    .groupby("cluster_id")
    .agg(
        num_segments=("segment", "count"),
        avg_char_length=("char_length", "mean"),
        std_char_length=("char_length", "std"),
        avg_herdan_c=("herdan_c", "mean"),
        std_herdan_c=("herdan_c", "std"),
        avg_mqm=("mqm_score", "mean"),
```

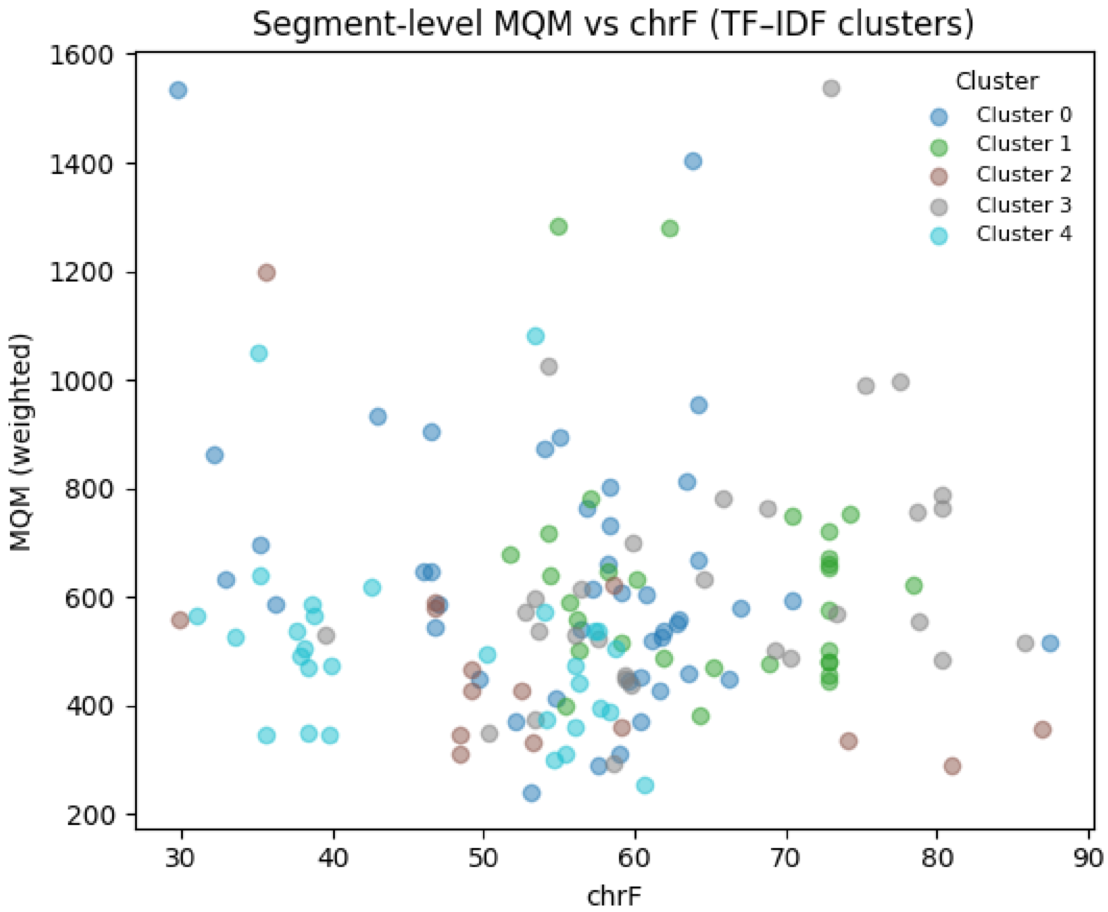
```
        avg_bleu=("BLEU", "mean"),
        avg_chrf=("chrF", "mean"),
        avg_comet=("COMET", "mean"),
    )
    .reset_index()
)

# Display cluster-level statistics
cluster_statistics
```

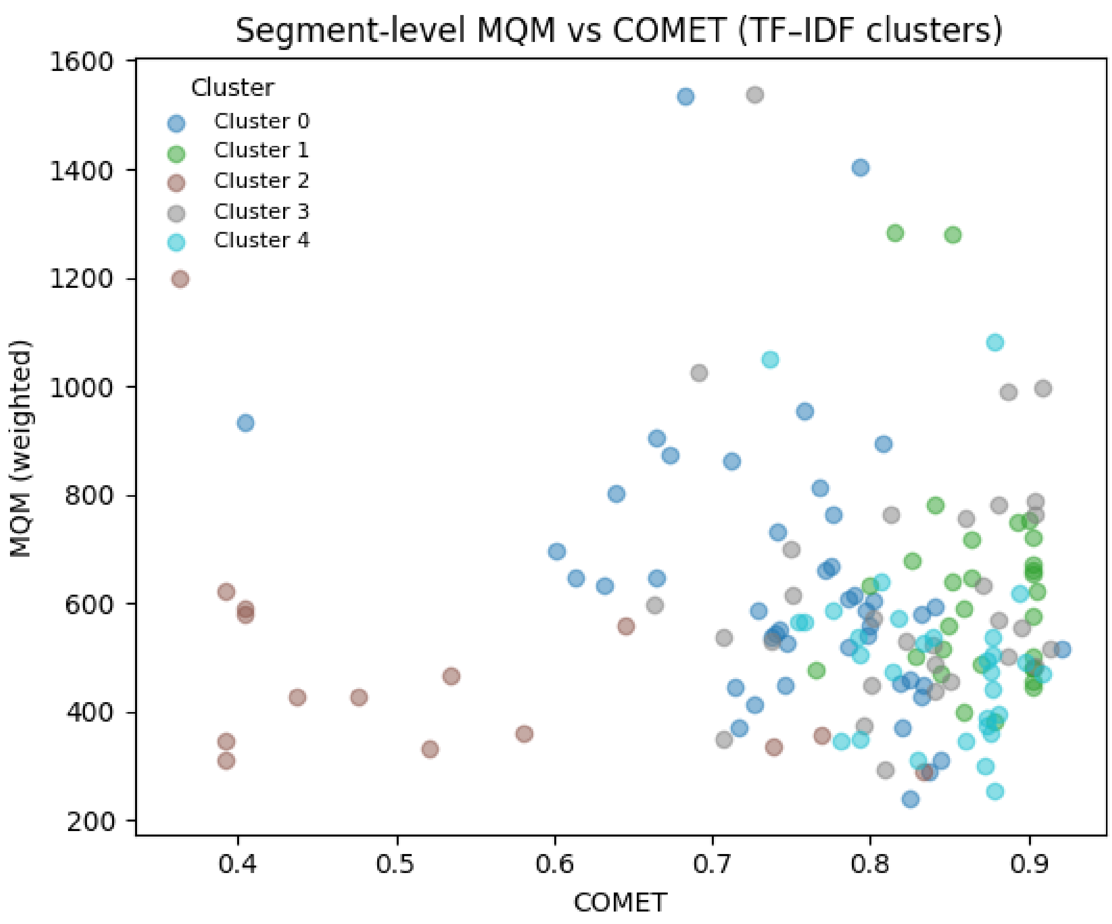
	cluster_id	num_segments	avg_char_length	std_char_length	avg_herdan_c	std_herdan_c	avg_
0	0	45	188.333333	114.347954	0.978628	0.022871	634.
1	1	30	152.000000	34.581239	0.983601	0.002260	627.
2	2	15	35.000000	0.000000	1.000000	0.000000	480.
3	3	30	207.233333	9.474770	0.972140	0.010851	637.
4	4	30	130.000000	62.042811	0.983785	0.016492	503.

```
import matplotlib.pyplot as plt

# Get TF-IDF cluster IDs
cluster_ids = sorted(segment_level_metrics_table["cluster_id"].unique())
color_map = plt.cm.get_cmap("tab10", len(cluster_ids))

# One plot per automatic metric, all TF-IDF clusters overlaid
for metric_name in ["BLEU", "chrF", "COMET"]:
    plt.figure(figsize=(6, 5))

    for idx, cluster_id in enumerate(cluster_ids):
        cluster_data = segment_level_metrics_table[
            segment_level_metrics_table["cluster_id"] == cluster_id
        ]

        plt.scatter(
            cluster_data[metric_name],
            cluster_data["mqm_score"],
            alpha=0.5,
            color=color_map(idx),
            label=f"Cluster {cluster_id}",
        )

    plt.xlabel(metric_name)
    plt.ylabel("MQM (weighted)")
    plt.title(f"Segment-level MQM vs {metric_name} (TF-IDF clusters)")
    plt.legend(
        title="Cluster",
        fontsize=8,
        title_fontsize=9,
        frameon=False,
    )

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

```
/tmp/ipython-input-1955519037.py:5: MatplotlibDeprecationWarning: The get_cmap function was deprecated in
↪  Matplotlib 3.7 and will be removed in 3.11. Use ``matplotlib.colormaps[name]`` or
↪  ``matplotlib.colormaps.get_cmap()`` or ``pyplot.get_cmap()`` instead.
  color_map = plt.cm.get_cmap("tab10", len(cluster_ids))
```


Segment-level MQM vs BLEU (TF–IDF clusters)
MQM (weighted)
BLEU
Cluster
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
0
10
20
30
40
50
60
70
80
200
400
600
800
1000
1200
1400
1600


Segment-level MQM vs chrF (TF–IDF clusters)
MQM (weighted)
chrF
Cluster
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
30
40
50
60
70
80
90
200
400
600
800
1000
1200
1400
1600



We can also try with embeddings-based clustering.

```
from sentence_transformers import SentenceTransformer

# Load a lightweight sentence embedding model
embedding_model = SentenceTransformer("all-MiniLM-L6-v2")

# Encode source sentences into dense vectors
source_sentences = segment_level_metrics_table["source"].astype(str).tolist()
sentence_embeddings = embedding_model.encode(
    source_sentences,
    batch_size=32,
    show_progress_bar=True,
)

# Cluster sentences in embedding space
num_clusters = 5
kmeans_model = KMeans(
    n_clusters=num_clusters,
    n_init=10,
    random_state=0,
)
embedding_cluster_ids = kmeans_model.fit_predict(sentence_embeddings)

# Append embedding-based cluster IDs to the segment-level table
segment_level_metrics_table["embedding_cluster"] = embedding_cluster_ids

embedding_cluster_correlation_results = []

for cluster_id in sorted(segment_level_metrics_table["embedding_cluster"].unique()):
    cluster_data = segment_level_metrics_table[
        segment_level_metrics_table["embedding_cluster"] == cluster_id
    ]

    for metric_name in ["BLEU", "chrF", "COMET"]:
        pearson_r, _ = pearsonr(
            cluster_data["mqm_score"],
            cluster_data[metric_name],
        )
        spearman_r, _ = spearmanr(
```

```
            cluster_data["mqm_score"],
            cluster_data[metric_name],
        )

        embedding_cluster_correlation_results.append({
            "embedding_cluster": cluster_id,
            "metric": metric_name,
            "num_segments": len(cluster_data),
            "pearson_r": pearson_r,
            "spearman_r": spearman_r,
        })

# Convert to DataFrame
embedding_cluster_correlation_table = pd.DataFrame(embedding_cluster_correlation_results)

embedding_cluster_correlation_table
```

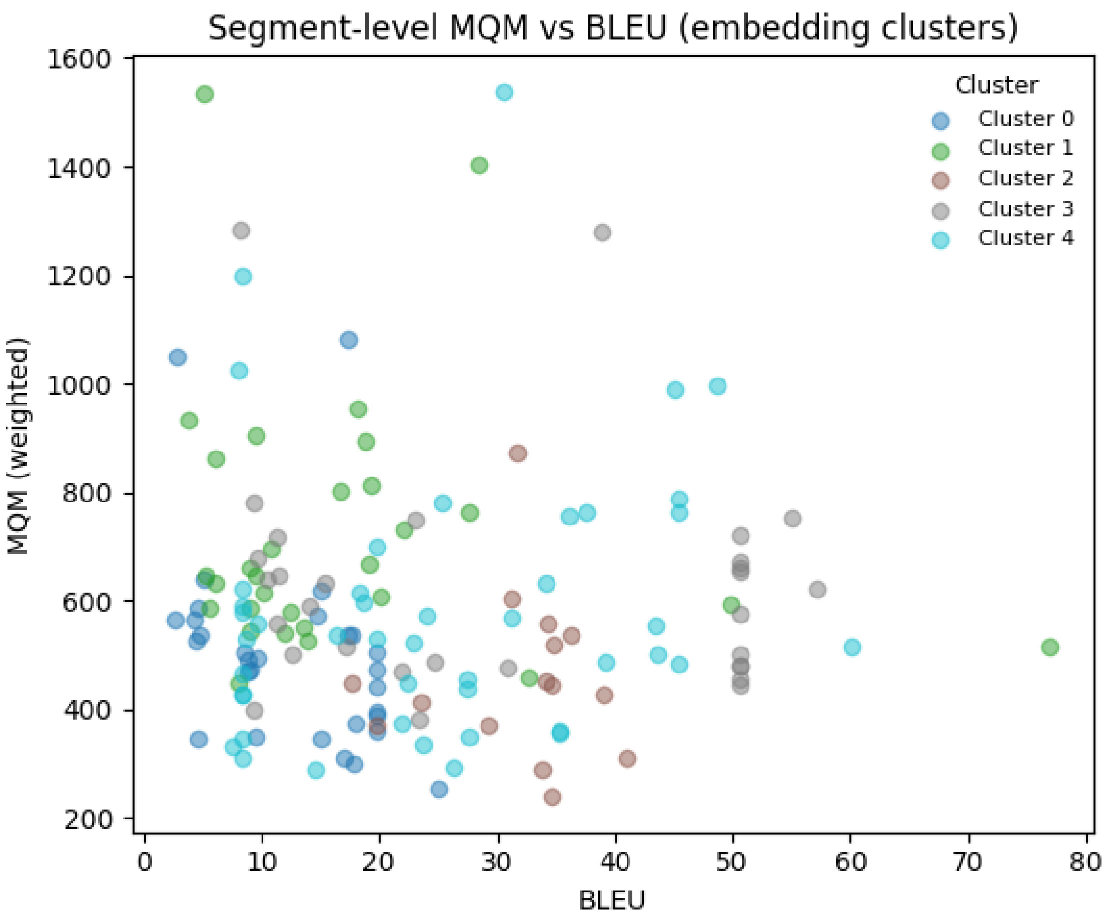
```
Batches:   0%|          | 0/5 [00:00<?, ?it/s]
```

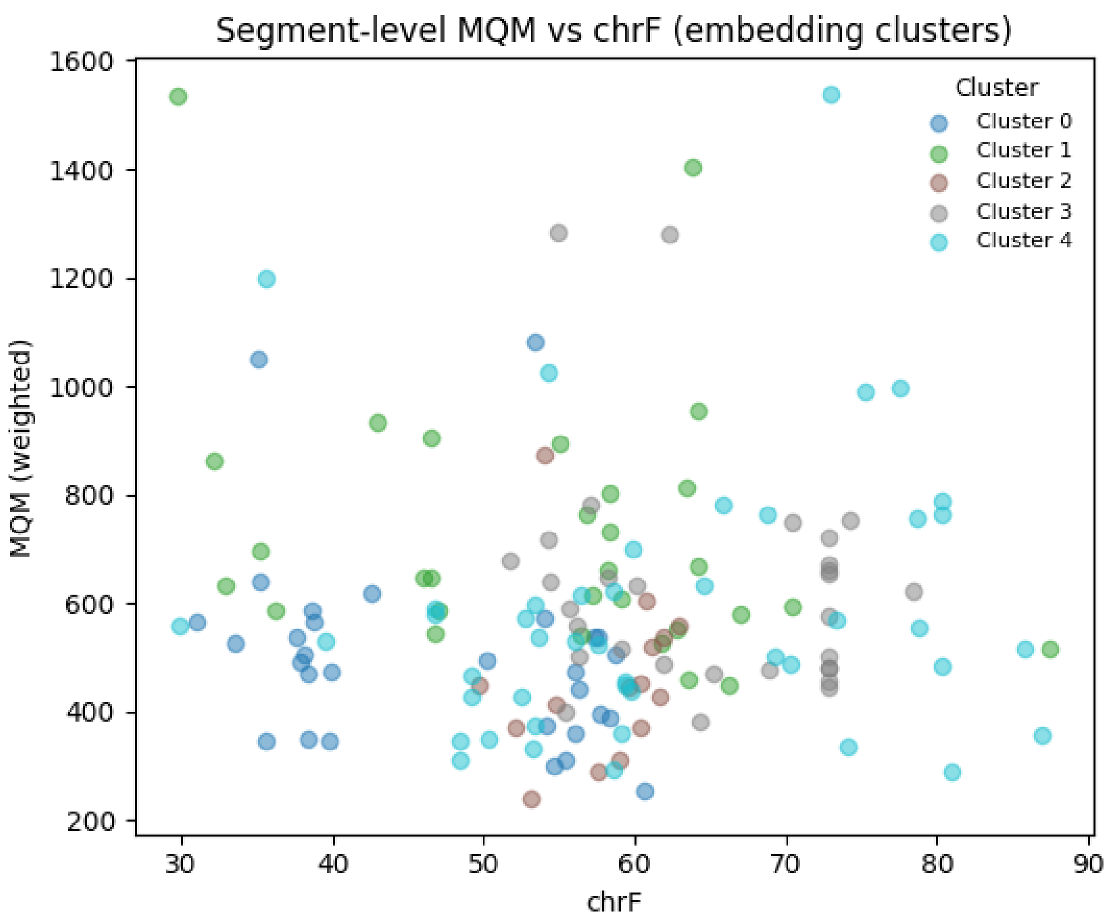
	embedding_cluster	metric	num_segments	pearson_r	spearman_r
0	0	BLEU	30	-0.350920	-0.477662
1	0	chrF	30	-0.264445	-0.367824
2	0	COMET	30	-0.292790	-0.235648
3	1	BLEU	30	-0.132901	-0.090565
4	1	chrF	30	-0.315341	-0.346646
5	1	COMET	30	-0.260122	-0.369785
6	2	BLEU	15	-0.016728	-0.025000
7	2	chrF	15	0.123890	0.425000
8	2	COMET	15	-0.571589	-0.496429
9	3	BLEU	30	-0.105968	-0.140210
10	3	chrF	30	-0.197769	-0.134996
11	3	COMET	30	-0.177681	-0.120941
12	4	BLEU	45	0.182905	0.209236
13	4	chrF	45	0.121521	0.149173
14	4	COMET	45	0.104267	0.216490

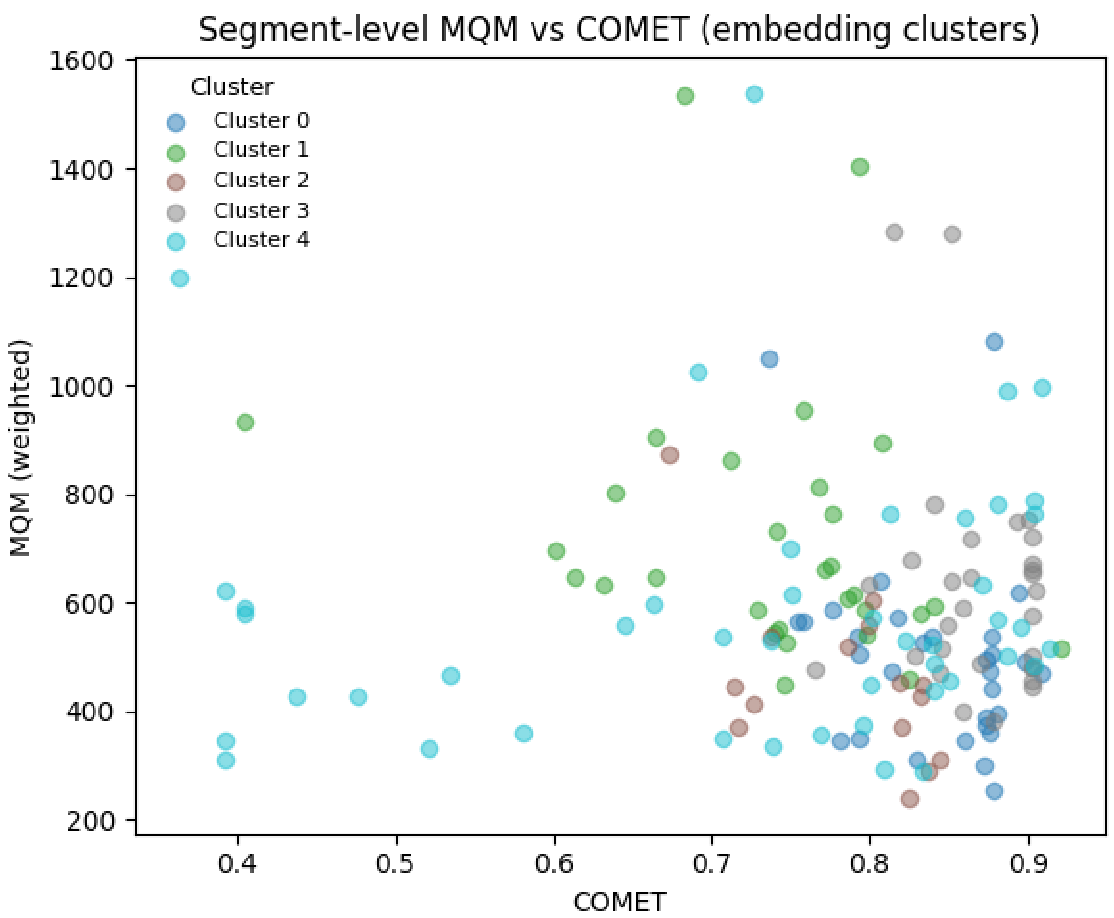
```
import matplotlib.pyplot as plt

# Get embedding cluster IDs
cluster_ids = sorted(segment_level_metrics_table["embedding_cluster"].unique())
color_map = plt.cm.get_cmap("tab10", len(cluster_ids))

# One plot per automatic metric, all clusters overlaid
for metric_name in ["BLEU", "chrF", "COMET"]:
    plt.figure(figsize=(6, 5))

    for idx, cluster_id in enumerate(cluster_ids):
        cluster_data = segment_level_metrics_table[
            segment_level_metrics_table["embedding_cluster"] == cluster_id
        ]

        plt.scatter(
            cluster_data[metric_name],
            cluster_data["mqm_score"],
            alpha=0.5,
            color=color_map(idx),
            label=f"Cluster {cluster_id}",
```

```
        )

    plt.xlabel(metric_name)
    plt.ylabel("MQM (weighted)")
    plt.title(f"Segment-level MQM vs {metric_name} (embedding clusters)")
    plt.legend(
        title="Cluster",
        fontsize=8,
        title_fontsize=9,
        frameon=False,
    )

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

```
/tmp/ipython-input-1973396413.py:5: MatplotlibDeprecationWarning: The get_cmap function was deprecated in
↪ Matplotlib 3.7 and will be removed in 3.11. Use ``matplotlib.colormaps[name]`` or
↪ ``matplotlib.colormaps.get_cmap()`` or ``pyplot.get_cmap()`` instead.
  color_map = plt.cm.get_cmap("tab10", len(cluster_ids))
```




Segment-level MQM vs chrF (embedding clusters)
MQM (weighted)
chrF
Cluster
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
1600
1400
1200
1000
800
600
400
200
30
40
50
60
70
80
90


Segment-level MQM vs COMET (embedding clusters)
MQM (weighted)
COMET
Cluster
Cluster 0
Cluster 1
Cluster 2
Cluster 3
Cluster 4
1600
1400
1200
1000
800
600
400
200
0.4
0.5
0.6
0.7
0.8
0.9