Imports and Login

```
!pip install unsloth evaluate jiwer rouge-score sacrebleu
```

Phonetic Noise Augmentation Logic

```
import random
import re
import unicodedata
import torch
import numpy as np
import pandas as pd
from datasets import load_dataset, Dataset
from unsloth import FastLanguageModel
from unsloth.trainer import SFTTrainer
from transformers import TrainingArguments
import evaluate
from tqdm import tqdm
from sklearn.metrics import precision_recall_fscore_support

PHONETIC_SWAPS = {
    "v": ["w"], "w": ["v"], "z": ["j"], "j": ["z"], "f": ["ph"], "ph":
["f"],
    "ee": ["i"], "i": ["ee"], "oo": ["u"], "u": ["oo"], "s": ["sh"],
"sh": ["s"], "y": ["i"]
}
SLANG_MAP = {"hai": ["h", "ey"], "kya": ["ky"], "nahi": ["nhi",
"nai"], "raha": ["rha"]}

def inject_phonetic_noise(text, p=0.3):

    if not isinstance(text, str): return text
    words = text.split()
    augmented_words = []

    for w in words:
        low_word = w.lower()

        # 1. Noise Mapping
        if low_word in SLANG_MAP and random.random() < p:
            low_word = random.choice(SLANG_MAP[low_word])

        # 2. Phonetic Swaps
        for char, variants in PHONETIC_SWAPS.items():
            if char in low_word and random.random() < 0.2:
                low_word = low_word.replace(char,
random.choice(variants))

        # 3. Middle Vowel Drop
```

```
        if len(low_word) > 4 and random.random() < 0.15:
            vowels = re.findall(r"[aeiou]", low_word[1:-1])
            if vowels:
                low_word = low_word.replace(random.choice(vowels), "",
1)

        # 4. Entity Capitalization Noise (ADDED)
        # Randomly capitalize to train model on proper nouns and
varied user inputs
        if random.random() < 0.2:
            low_word = low_word.capitalize()
        elif random.random() < 0.05:
            low_word = low_word.upper()

        augmented_words.append(low_word)

    return " ".join(augmented_words)
```

Data Loading & Complex Character Normalization

```
# Load Dataset
ds = load_dataset("sk-community/romanized_hindi", split="train")
df = ds.to_pandas().head(3000)

def clean_and_normalize(row):
    if not isinstance(row["Hindi"], str) or not
isinstance(row["Transliterated Hindi"], str):
        return None
    # NFC merges decomposed characters (critical for half-letters
like)
    hindi = unicodedata.normalize('NFC', row["Hindi"].strip())
    hindi = re.sub(r"[^\u0900-\u097F0-9\s!?]", "", hindi)
    roman = row["Transliterated Hindi"].strip()
    return {"roman": roman, "hindi": hindi}

df = df.apply(clean_and_normalize,
axis=1).dropna().apply(pd.Series).reset_index(drop=True)

# Generate multi-variants to increase dataset scale and entity
robustness
df_noisy = df.copy()
df_noisy['roman'] = df_noisy['roman'].apply(lambda x:
inject_phonetic_noise(x))
df_final = pd.concat([df,
df_noisy]).drop_duplicates().reset_index(drop=True)
print(f"Dataset expanded to {len(df_final)} rows.")
```

{"model_id":"3fb3ef61fc924472aca04923cbbb5ab4","version_major":2,"version_minor":0}

{"model_id":"71306b4a3fef418eb9a062d7d623b965","version_major":2,"version_minor":0}

{"model_id":"9ca8050705c94a08a1eb6005c2bc914d","version_major":2,"version_minor":0}

{"model_id":"0e9fc14f13274071b9d9f71de250e2e0","version_major":2,"version_minor":0}

{"model_id":"e5ff87604a3e495da553640c8202e267","version_major":2,"version_minor":0}

{"model_id":"5601482a9e8f49d688213696308db837","version_major":2,"version_minor":0}

```
Dataset expanded to 5941 rows.
```

Model Setup and Phonetic Tokenization

```
from unsloth import FastLanguageModel

BASE_MODEL = "google/gemma-2b-it"
model, tokenizer = FastLanguageModel.from_pretrained(
    BASE_MODEL,
    max_seq_length=512,
    load_in_4bit=True
)

model = FastLanguageModel.get_peft_model(
    model, r=128, lora_alpha=128,
    target_modules=["q_proj", "k_proj", "v_proj", "o_proj",
"gate_proj", "up_proj", "down_proj"]
)

def tokenize(batch):
    input_ids, labels = [], []
    for r, h in zip(batch["roman"], batch["hindi"]):
        prompt_text = f"Hinglish: {r}\nHindi: "
        full_text = prompt_text + h + tokenizer.eos_token

        tokenized_full = tokenizer(full_text, truncation=True,
max_length=128)
        tokenized_prompt = tokenizer(prompt_text, truncation=True,
max_length=128, add_special_tokens=False)

        prompt_len = len(tokenized_prompt["input_ids"])
        # Mask prompt so model only learns to predict Devanagari
sequence
        label = [-100] * prompt_len + tokenized_full["input_ids"]
[prompt_len:]
```

```
        input_ids.append(tokenized_full["input_ids"])
        labels.append(label)
    return {"input_ids": input_ids, "labels": labels}

# Prepare final training data
dataset = Dataset.from_pandas(df_final).shuffle(seed=42)
split = dataset.train_test_split(test_size=0.1)
train_ds = split["train"].map(tokenize, batched=True,
remove_columns=dataset.column_names)
```

```
==((====))==  Unsloth 2026.1.2: Fast Gemma patching. Transformers:
4.57.3.
   \\   /|    Tesla T4. Num GPUs = 1. Max memory: 14.741 GB. Platform:
Linux.
O^O/ \_/ \    Torch: 2.9.1+cu128. CUDA: 7.5. CUDA Toolkit: 12.8.
Triton: 3.5.1
\        /    Bfloat16 = FALSE. FA [Xformers = 0.0.33.post2. FA2 =
False]
 "-____-"     Free license: http://github.com/unslothai/unsloth
Unsloth: Fast downloading is enabled - ignore downloading bars which
are red colored!
```


```
{"model_id":"85fd405b19c94c1296d7f3e4e8a03096","version_major":2,"vers
ion_minor":0}

{"model_id":"e181a7b8f7fb4b349f5b2531a34353f4","version_major":2,"vers
ion_minor":0}

{"model_id":"a86c329ba88844a18d5f44863c40102f","version_major":2,"vers
ion_minor":0}

{"model_id":"2033edfcebad40798fb94260d04aae2d","version_major":2,"vers
ion_minor":0}

{"model_id":"a81cf9aed50f4bfd89dcc9598f1a1c44","version_major":2,"vers
ion_minor":0}

{"model_id":"21169ac2b983443298ae67e1d1bc9393","version_major":2,"vers
ion_minor":0}
```


```
Unsloth 2026.1.2 patched 18 layers with 18 QKV layers, 18 O layers and
18 MLP layers.
```


```
{"model_id":"d5f97d95df754212953e366d9fdb49fd","version_major":2,"vers
ion_minor":0}
```


Training and Evaluation

```
from unsloth.trainer import SFTTrainer
from transformers import TrainingArguments
```

```
# Trainer setup
trainer = SFTTrainer(
    model=model, tokenizer=tokenizer, train_dataset=train_ds,
    args=TrainingArguments(
        output_dir="./hinglish_lora", per_device_train_batch_size=4,
        gradient_accumulation_steps=4, learning_rate=5e-5,
num_train_epochs=2,
        fp16=True, logging_steps=50, optim="adamw_8bit",
report_to="none"
    ),
)
trainer.train()
```

```
The model is already on multiple devices. Skipping the move to device
specified in `args`.
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs used = 1
   \\   /|    Num examples = 5,346 | Num Epochs = 2 | Total steps =
670
O^O/ \_/ \    Batch size per device = 4 | Gradient accumulation steps
= 4
\        /    Data Parallel GPUs = 1 | Total batch size (4 x 4 x 1) =
16
 "-____-"     Trainable parameters = 156,893,184 of 2,663,065,600
(5.89% trained)

<IPython.core.display.HTML object>

TrainOutput(global_step=670, training_loss=0.4612732574121276,
metrics={'train_runtime': 1508.0428, 'train_samples_per_second': 7.09,
'train_steps_per_second': 0.444, 'total_flos': 1.49746428112896e+16,
'train_loss': 0.4612732574121276, 'epoch': 2.0})
```

Metric and Evaluation Utility Setup

```
import torch

def generate_text_fixed(roman):

    # Force lowercase as models often handle standardized case better
for phonetics
    prompt = f"Hinglish: {roman.lower()}\nHindi: "
    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)

    with torch.no_grad():
        out = model.generate(
            **inputs,
            max_new_tokens=128,
            do_sample=False,
            repetition_penalty=1.2,
            no_repeat_ngram_size=3,
```

```
            pad_token_id=tokenizer.eos_token_id
        )

    # Extract newly generated tokens only
    input_len = inputs.input_ids.shape[1]
    decoded = tokenizer.decode(out[0][input_len:],
skip_special_tokens=True).strip()
    return decoded

import evaluate
from sklearn.metrics import precision_recall_fscore_support
import unicodedata

def compute_all_metrics(preds, refs):


    # 1. Standard character and sequence metrics
    cer = evaluate.load("cer").compute(predictions=preds,
references=refs)
    wer = evaluate.load("wer").compute(predictions=preds,
references=refs)
    chrf = evaluate.load("chrf").compute(predictions=preds,
references=refs)["score"]

    # 2. FLATTEN sentences into words for Word-Level Accuracy
    all_pred_words = []
    all_ref_words = []

    for p_sent, r_sent in zip(preds, refs):
        # Splitting sentences into individual words
        p_words = p_sent.split()
        r_words = r_sent.split()

        # Alignment padding: Ensure we compare equal lengths
        max_len = max(len(p_words), len(r_words))
        p_words += ["<PAD>"] * (max_len - len(p_words))
        r_words += ["<PAD>"] * (max_len - len(r_words))

        all_pred_words.extend(p_words)
        all_ref_words.extend(r_words)

    # 3. Calculate binary success for every individual word
    y_true = [1] * len(all_ref_words)
    y_pred = [1 if p == r else 0 for p, r in zip(all_pred_words,
all_ref_words)]

    # Compute Word-Level metrics
    prec, rec, f1, _ = precision_recall_fscore_support(
        y_true, y_pred, average='binary', zero_division=0
    )
```

```
    return {
        "CER": cer,
        "WER": wer,
        "chrF_Score": chrf,
        "Model_Word_Precision": prec,
        "Model_Word_Recall": rec,
        "Model_Word_F1": f1
    }
```

Evaluation on Internal Dataset

```
print("--- Internal Dataset Evaluation (Word-Level Recall/F1) ---")

# Ensure evaluation subset is selected
eval_subset_internal =
split["test"].shuffle(seed=42).select(range(min(50,
len(split["test"]))))

preds_int, refs_int = [], []
for ex in tqdm(eval_subset_internal, desc="Internal Eval"):
    # Using your fixed prediction logic (Greedy + Repetition Penalty)
    p = generate_text_fixed(ex["roman"])
    # Normalize to NFC to ensure character clusters match correctly
    r = unicodedata.normalize('NFC', ex["hindi"].strip())

    preds_int.append(p)
    refs_int.append(r)

# Compute new Model Recall and F1 metrics
internal_results = compute_all_metrics(preds_int, refs_int)

for metric, value in internal_results.items():
    print(f"{metric}: {value:.4f}")
```

```
--- Internal Dataset Evaluation (Word-Level Recall/F1) ---

Internal Eval: 100%|██████████| 50/50 [01:36<00:00,  1.93s/it]

CER: 0.1694
WER: 0.3011
chrF_Score: 71.9108
Model_Word_Precision: 1.0000
Model_Word_Recall: 0.5282
Model_Word_F1: 0.6913
```

Evaluation on External codebyam Dataset

```
from datasets import load_dataset
import unicodedata
from tqdm import tqdm
```

```python
print("\n--- External Dataset Evaluation (codebyam - 50 rows) ---")

# Load the external dataset
ext_ds_full = load_dataset("codebyam/Hinglish-Hindi-Transliteration-
Dataset", split="train")

# Select a subset of 50 rows for evaluation
eval_subset_ext = ext_ds_full.shuffle(seed=42).select(range(50))

preds_ext, refs_ext = [], []

for ex in tqdm(eval_subset_ext, desc="External Eval"):
    # Generate transliteration using your fixed prediction function
    # (Greedy search + repetition penalty for high precision)
    p = generate_text_fixed(ex["Hinglish"])

    # Normalize Devanagari to NFC to ensure complex phonetic clusters
    # (like half-characters) are compared accurately
    r = unicodedata.normalize('NFC', ex["Hindi"].strip())

    preds_ext.append(p)
    refs_ext.append(r)

# Compute word-level metrics: CER, WER, chrF, Model Recall, and Model
F1
external_results = compute_all_metrics(preds_ext, refs_ext)

for metric, value in external_results.items():
    print(f"{metric}: {value:.4f}")
```

```
--- External Dataset Evaluation (codebyam - 50 rows) ---

External Eval: 100%|██████████| 50/50 [00:41<00:00,  1.21it/s]

CER: 0.1978
WER: 0.3370
chrF_Score: 62.6390
Model_Word_Precision: 1.0000
Model_Word_Recall: 0.6258
Model_Word_F1: 0.7699
```

Data Samples

```python
print("--- Testing on 5 Samples from External Dataset (codebyam) ---")

# Create a subset of 5 samples from the external dataset
eval_subset_ext_5_samples =
ext_ds_full.shuffle(seed=42).select(range(min(5, len(ext_ds_full))))
```

```python
preds_ext_5_samples, refs_ext_5_samples = [], []

for i, ex in enumerate(eval_subset_ext_5_samples):
    # Generate text using sampling logic
    p = generate_text_fixed(ex["Hinglish"])
    # Normalize external references for a fair comparison of phonetic
clusters
    r = unicodedata.normalize('NFC', ex["Hindi"].strip())

    preds_ext_5_samples.append(p)
    refs_ext_5_samples.append(r)

    print(f"\nSample {i+1}:")
    print(f"Roman (Input): {ex['Hinglish']}")
    print(f"Predicted Hindi: {p}")
    print(f"Actual Hindi:    {r}")
```

```
--- Testing on 5 Samples from External Dataset (codebyam) ---

Sample 1:
Roman (Input): ab mujhe sirf ek cheez chahiye — sukoon, jo tumhare
saath tha.
Predicted Hindi: अब मुझे सिर्फ एक चीज चाहिए सुखन जो तुम्हारे साथ था।
Actual Hindi:    अब मुझे सिर्फ़ एक चीज़ चाहिए – सुकून, जो तुम्हारे साथ था|

Sample 2:
Roman (Input): kal office band hai kya?
Predicted Hindi: कैल कार्यालय बन्द है क्या?
Actual Hindi:    कल ऑफिस बंद है क्या?

Sample 3:
Roman (Input): birthday ke din to sab friend ek hi line bolte the -
treat chahiye.
Predicted Hindi: न्मदिन के दिन तो सब फ्रेंड एक ही लाइन बोले थे  ट्रेट चाहिए।
Actual Hindi:    बर्थडे के दिन तो सब फ्रेंड एक ही लाइन बोलते थे -  ट्रीट चाहिए|

Sample 4:
Roman (Input): humari dosti kabhi nahi tootegi
Predicted Hindi: हॉरी दोस्ती कभी नहीं टूटेंगे
Actual Hindi:    हमारी दोस्ती कभी नहीं टूटेगी

Sample 5:
Roman (Input): kya tumhara car ka insurance renew ho gaya?
Predicted Hindi: क्या तुमरा कार का इन्स्ुरेंस रिवन हो गया?
Actual Hindi:    क्या तुम्हारा कार का इंश्योरेंस रीन्यू हो गया?
```

```python
print("--- Testing on 5 Samples from Internal Test Split ---")

# Create a subset of 5 samples from the internal test split
eval_subset_5_samples =
```

```
split["test"].shuffle(seed=42).select(range(min(5,
len(split["test"]))))

preds_5_samples, refs_5_samples = [], []

for i, ex in enumerate(eval_subset_5_samples):
    # Generate text using sampling logic
    p = generate_text_fixed(ex["roman"])
    # Normalize Devanagari for consistent character clustering
    r = unicodedata.normalize('NFC', ex["hindi"].strip())

    preds_5_samples.append(p)
    refs_5_samples.append(r)

    print(f"\nSample {i+1}:")
    print(f"Roman (Input): {ex['roman']}")
    print(f"Predicted Hindi: {p}")
    print(f"Actual Hindi:    {r}")
```

```
--- Testing on 5 Samples from Internal Test Split ---

Sample 1:
Roman (Input): mgarba asrganj, munger, bihar sthit ek gaaon hai.
Predicted Hindi: मग़राबासारंजण मुंगेर बिहार स्थित एक गाँव है।
Actual Hindi:    मंगरबा असरगंज मुंगेर बिहार स्थित एक गाँव है।

Sample 2:
Roman (Input): 40-65 aiu adheektar logo ke beech dooriian lata hai.
Predicted Hindi: ४०६५ आधीं एकता के बीच दूरीयाँ लता है।
Actual Hindi:    ४०६५ आयु अधिकतर लोगो के बीच दुरियँ लाता है।

Sample 3:
Roman (Input): Dep ke atirikt vcharon Men eensanee anguthn se banee ek
nekles hai jise shpairo us samai pahnata Ey jab pailegoshto oose khane
ke liye taiaar hota Hai, aur rjdand dep ke Ek dosht ke paas mauzud
raajdand par Aadharit Thi.
Predicted Hindi: देप के अतिरिक्त वर्चसों में एकनिष्ठ अंगूठे से बनी एक नक़्ले है जिसे स्पैरो उस same
की पहनना के लिए तैयार होता है और रजदंड देप के एक दोस्त के पास मौजूद राजदंड पर आधारित था।
Actual Hindi:    डेप के अतिरिक्त विचारों में इंसानी अंगूठों से बनी एक नेकलेस है जिसे स्पैरो उस
समय पहनता है जब पैलेगोस्टो उसे खाने के लिए तैयार होता है और राजदंड डेप के एक दोस्त के पास मौजूद
राजदंड पर आधारित थी।

Sample 4:
Roman (Input): unhe sham hoot Ke saath - ek doctor ko bhoomeegat
Sangeet drushy men achhee TARAH sthapit the - formntara bheja gaya,
lekin inmen sdhar Ke Koi Sanket naee dkhe.
Predicted Hindi: उनसे सामान्यतः एक डॉक्टर को भूमीगत संगीत दृश्य में अच्छी तरह स्थापित थे
फॉर्मंडल को बेजबा गया लेकिन इनमें से कोई संकेत नहीं दिखे।
Actual Hindi:    उन्हें सैम हट के साथ  एक डॉक्टर जो भूमिगत संगीत दृश्य में अच्छी तरह स्थापित थे
```

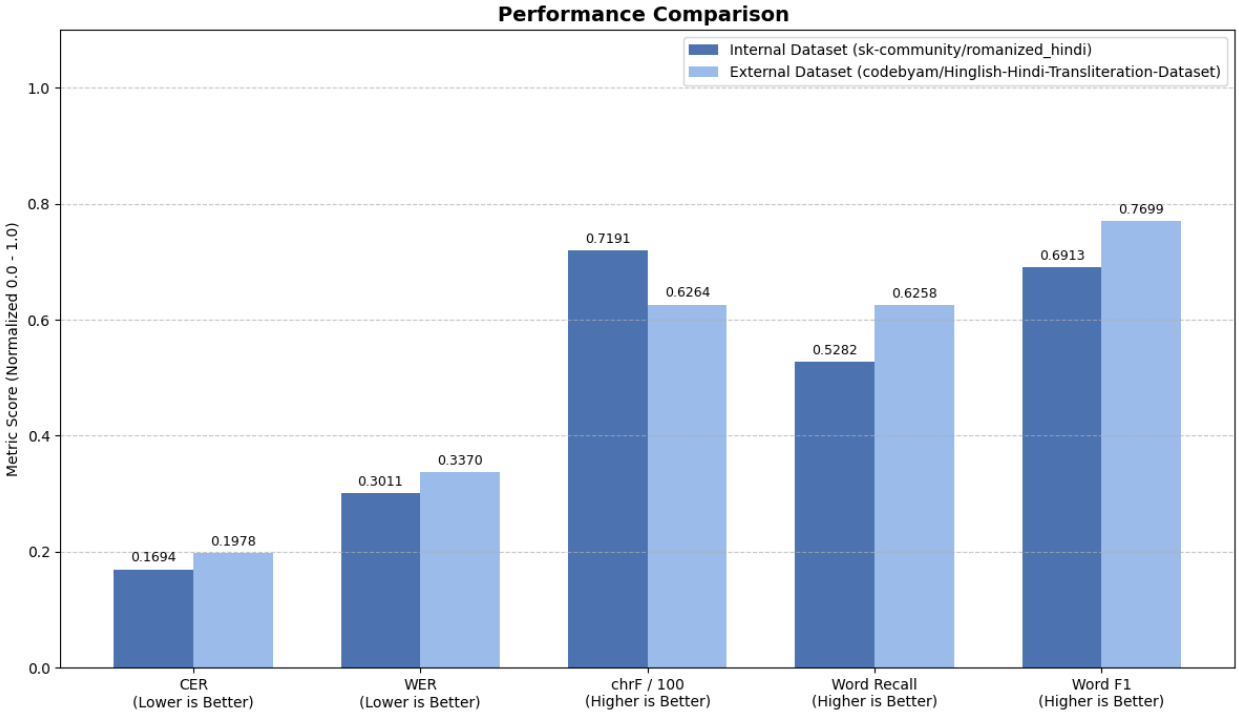
फोर्मेंटरा भेजा गया लेकिन उनमें सुधार के कोई संकेत नहीं दिखे

```
Sample 5:
Roman (Input): Rashtriya Aerospace Prayaogshala
Predicted Hindi: राष्ट्रीय एयरस्पेस प्रयोगशाला
Actual Hindi:    राष्ट्रीय एयरोस्पेस प्रयोगशाला
```

Performance Comparison

```
import matplotlib.pyplot as plt
import numpy as np

# Define the metrics to be plotted
plot_metrics = [
    "CER\n(Lower is Better)",
    "WER\n(Lower is Better)",
    "chrF / 100\n(Higher is Better)",
    "Word Recall\n(Higher is Better)",
    "Word F1\n(Higher is Better)"
]

# Real-time data extraction from your evaluation dictionaries
# This will automatically update whenever you re-run your evaluation
blocks
try:
    internal_plot_values = [
        internal_results["CER"],
        internal_results["WER"],
        internal_results["chrF_Score"] / 100, # Normalized to 0.0-1.0
scale
        internal_results["Model_Word_Recall"],
        internal_results["Model_Word_F1"]
    ]

    external_plot_values = [
        external_results["CER"],
        external_results["WER"],
        external_results["chrF_Score"] / 100, # Normalized to 0.0-1.0
scale
        external_results["Model_Word_Recall"],
        external_results["Model_Word_F1"]
    ]
except KeyError as e:
    print(f"Error: Metric {e} not found. Ensure compute_all_metrics
has run for both datasets.")
    internal_plot_values = [0]*5
    external_plot_values = [0]*5

x = np.arange(len(plot_metrics))
```

```
width = 0.35

fig, ax = plt.subplots(figsize=(12, 7))
rects1 = ax.bar(x - width/2, internal_plot_values, width,
label='Internal Dataset (sk-community/romanized_hindi)',
color='#4C72B0')
rects2 = ax.bar(x + width/2, external_plot_values, width,
label='External Dataset (codebyam/Hinglish-Hindi-Transliteration-
Dataset)', color='#9BBBEA')

# Styling and dynamic labeling
ax.set_ylabel('Metric Score (Normalized 0.0 - 1.0)')
ax.set_title('Performance Comparison ', fontsize=14,
fontweight='bold')
ax.set_xticks(x)
ax.set_xticklabels(plot_metrics, fontsize=10)
ax.legend()
ax.grid(axis='y', linestyle='--', alpha=0.7)

# Dynamic value labels on top of bars
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate(f'{height:.4f}',
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),
                    textcoords="offset points",
                    ha='center', va='bottom', fontsize=9)

autolabel(rects1)
autolabel(rects2)

plt.ylim(0, 1.1) # Limits set for visibility of top-labels
plt.tight_layout()
plt.show()
```


Performance Comparison
Internal Dataset (sk-community/romanized_hindi)
External Dataset (codebyam/Hinglish-Hindi-Transliteration-Dataset)
Metric Score (Normalized 0.0 - 1.0)
0.0
0.2
0.4
0.6
0.8
1.0
0.1694
0.1978
0.3011
0.3370
0.7191
0.6264
0.5282
0.6258
0.6913
0.7699
CER
(Lower is Better)
WER
(Lower is Better)
chrF / 100
(Higher is Better)
Word Recall
(Higher is Better)
Word F1
(Higher is Better)