Import and Install all Liberaries

```
%%capture
!pip install transformers
!python --version
!pip install nltk rouge-score bert_score

from transformers import AutoModelForCausalLM, AutoTokenizer
import torch
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import (
    mean_absolute_error,
    mean_squared_error,
    r2_score,
    mean_absolute_percentage_error,
    median_absolute_error
)
import matplotlib.pyplot as plt
from matplotlib.colors import rgb2hex
from scipy.stats import spearmanr, kendalltau, pearsonr
import json
from tqdm import tqdm
import nltk
from huggingface_hub import notebook_login
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import difflib
import nltk
from nltk.translate.bleu_score import sentence_bleu, SmoothingFunction
from nltk.translate.meteor_score import meteor_score
from rouge_score import rouge_scorer
from bert_score import score as bertscore
import re
import os

hf_token = "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"
#"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"
HF_TOKEN = hf_token

# add token
os.environ['HF_TOKEN'] = hf_token
notebook_login(hf_token)
nltk.download('punkt')
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/
_deprecation.py:38: FutureWarning: Deprecated positional argument(s)
used in 'notebook_login': pass
new_session='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' as keyword args.
From version 1.0 passing these as positional arguments will result in
an error,
  warnings.warn(
```


{"model_id":"fb8d83e568114c6a84295ca6734d7e5e","version_major":2,"version_minor":0}


```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

```
True
```

Evaluate Checking Concepts

ALL Models Here

Smol Model

```python
# Constants
MODEL_CHECKPOINT = "HuggingFaceTB/SmolLM2-1.7B-Instruct"
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")

SYS_CHECK_PROMPT = """
You are a Python expert and debugging assistant.
Your task is to evaluate the provided Python code against a list of
concepts.
For each concept, verify both its syntactic and logical correctness.
Output exactly one line per concept in the following format:
"<concept number>) yes" or "<concept number>) no"
Do not include any additional commentary, explanations, or error
details.
Ensure the number of output lines matches the number of concepts
provided, with no extra spaces or characters.
"""

CHECK_PROMPT = """
Evaluate whether the student's Python code satisfies each of the
following concepts:

{concepts}

Student's Code:

{code}
```

```
Provide your evaluation using the exact format specified in the system
prompt.
"""

class SmolLM:
    def __init__(self, checkpoint):
        self.tokenizer = AutoTokenizer.from_pretrained(checkpoint)
        self.model =
AutoModelForCausalLM.from_pretrained(checkpoint).to(DEVICE)

    def apply_chat_template(self, sys_prompt, user_prompt):
        messages = [
            {"role": "system", "content": sys_prompt},
            {"role": "user", "content": user_prompt}
        ]
        return self.tokenizer.apply_chat_template(messages,
tokenize=False)

    def check_concepts(self, concepts, code, temperature= 0.9,
max_new_tokens= 1024, top_p= 0.94, do_sample = True):
        # Prepare the user prompt
        user_prompt = CHECK_PROMPT.format(concepts=concepts,
code=code)

        # Prepare input using chat template
        in_text = self.apply_chat_template(SYS_CHECK_PROMPT,
user_prompt)
        inputs = self.tokenizer.encode(in_text,
return_tensors="pt").to(DEVICE)

        # Generate response
        outputs = self.model.generate(
            inputs,
            max_new_tokens=max_new_tokens,
            temperature=temperature,
            top_p=top_p,
            do_sample=do_sample,
        )

        str_output = self.tokenizer.decode(outputs[0],
skip_special_tokens=True)

        output =str_output.split("\nassistant")[-1].strip()

        return output
```

Gemma Model

```
from huggingface_hub import InferenceClient

GEMMA_CHECK_PROMPT = """
```

```
You are a Python expert. Your task is to evaluate the following Python
code against a list of concepts.

Student's Code:
{code}

Concepts to Evaluate:
{concepts}

Instructions:
1. For each concept, check both the syntactic and logical correctness
of the code.
2. For each concept, output exactly one line in the format:
   "<concept number>) yes" or "<concept number>) no"
3. Do not provide any explanations or additional text.
4. Your final output must have exactly one line per concept, in the
correct order, with no extra spaces or characters.

Output:
Provide your evaluation in the exact format specified above.
"""

GEMMA_COMPARE_PROMPT = """
You are an AI writing assistant with deep expertise in Python
programming and code debugging.
Carefully analyze and strictly evaluate the Python codes provided by
the user and compare them.
Your output should be a single numerical score representing the
similarity between the two codes.
Do not include any text or formatting besides the score, which should
be between 0 and 100.
Do NOT repeat the system or user prompts in your final response.

help me with a score between 0 and 100 that reflects how similar the
professor's code is to the student's code:

Professor's code:
{pr_code}

Student's code:
{st_code}

Instructions:
1. Return just a number between 0 and 100.
"""

class Gemma:
    def __init__(self, model):
        self.client = InferenceClient(api_key=hf_token)
        self.model = model
```

```
    def check_concepts(self, concepts, code, max_tokens=200):
        prompt = GEMMA_CHECK_PROMPT.format(concepts=concepts,
code=code)

        messages = [
            {"role": "user", "content": prompt},
        ]

        response = self.client.chat.completions.create(
            model=self.model,
            messages=messages,
            max_tokens=max_tokens,
            stream=False
        )
        try:
            return response["choices"][0]["message"]
["content"].strip()
        except (KeyError, TypeError):
            return str(response).strip()

    def compare_code(self, pr_code, st_code, max_tokens=5):
        prompt = GEMMA_COMPARE_PROMPT.format(pr_code=pr_code,
st_code=st_code)

        messages = [
            {"role": "user", "content": prompt},
        ]

        response = self.client.chat.completions.create(
            model=self.model,
            messages=messages,
            max_tokens=max_tokens,
            stream=False,
        )
        try:
            result = response["choices"][0]["message"]
["content"].strip()
        except (KeyError, TypeError):
            result = str(response).strip()

        try:
            score = int(result)
            score = max(0, min(100, score))
        except ValueError:
            print("Invalid score format:", result)
            score = 0

        return score
```

```python
# test
# "google/gemma-2-2b-it"
# "google/gemma-2-9b-it"
# "google/gemma-2-27b-it"

gemma = Gemma("google/gemma-2-27b-it")

# Example for check_concepts
concepts = """
1) Proper class definition with __init__ method.
2) Use of encapsulation (private attributes).
3) Correct method definitions.
4) Use of classmethod.
"""
student_code = """
class Animal:
    \"\"\"This class represents a general animal.\"\"\"

    def __init__(self, name, sound):
        \"\"\"Initializes an Animal object with name and sound.\"\"\"
        self.__name = name
        self.__sound = sound

    def get_name(self):
        \"\"\"Returns the animal's name.\"\"\"
        return self.__name

    def make_sound(self):
        \"\"\"Makes the animal sound.\"\"\"
        print(f"{self.__name} says: {self.__sound}")

    @classmethod
    def create_from_name(cls, name):
        \"\"\"Creates an Animal object from a name.\"\"\"
        return cls(name, "Generic Sound")
"""
print("Check Concepts Output:")
print(gemma.check_concepts(concepts, student_code))

# Example for compare_code
professor_code = """
class Animal:
    def __init__(self, name, sound):
        self.name = name
        self.sound = sound
    def make_sound(self):
        print(f"{self.name} makes a sound: {self.sound}")
"""
score = gemma.compare_code(professor_code, student_code)
```

```
print("\nComparison Score:")
print(score)
```

```
Check Concepts Output:
1) yes
2) yes
3) yes
4) yes

Comparison Score:
50
```

Mistral Model

```
import requests

MISTRAL_CHECK_PROMPT = """
<s>
You are a Python expert. Check if the student's code satisfies the
following concepts:

{concepts}

Student's Code:

{code}

</s>
[INST]
For each concept, output exactly one line in the format:
"<concept number>) yes" or "<concept number>) no"

Do not include additional commentary or text.
[/INST]
"""

MISTRAL_COMPARE_PROMPT = """
<s>
You are a Python expert. help me with a score between 0 and 100 that
reflects how similar the the first code is to the second code:

First Code:
{pr_code}

Second Code:
{st_code}
</s>
[INST]
Return EXACTLY ONE integer between 0 and 100. Do not include any extra
text, punctuation, or formatting. If you are uncertain, output 0.
[/INST]
```

```
"""


class Mistral:
    def __init__(self, api_url):
        self.api_url = api_url
        self.hf_token = HF_TOKEN

        if not self.api_url or not self.hf_token:
            raise ValueError("Both API_URL and TOKEN must be provided
either as parameters or via environment variables.")
        self.headers = {"Authorization": f"Bearer {self.hf_token}"}

    def _query(self, payload):
        response = requests.post(self.api_url, headers=self.headers,
json=payload)
        response.raise_for_status()
        return response.json()

    def check_concepts(self, concepts, code, parameters=None):
        prompt = MISTRAL_CHECK_PROMPT.format(concepts=concepts,
code=code)

        if parameters is None:
            parameters = {
                "temperature": 1.0,
                "max_length": 1024,
                "top_p": 0.9,
                "top_k": 50,
                "repetition_penalty": 1.1
            }

        payload = {
            "inputs": prompt,
            "parameters": parameters
        }
        result = self._query(payload)
        try:
            generated_text = result[0]["generated_text"]
        except (IndexError, KeyError, TypeError):
            generated_text = str(result)

        feedback = generated_text.strip().split("[/INST]")[-1]
        return feedback.strip()

    def compare_code(self, pr_code, st_code, parameters=None):
        prompt = MISTRAL_COMPARE_PROMPT.format(pr_code=pr_code,
st_code=st_code)

        if parameters is None:
```

```
            parameters = {
                "temperature": 0.2,
                "max_length": 4,
                "top_p": 0.9,
                "top_k": 10,
                "repetition_penalty": 1.1
            }

        payload = {
            "inputs": prompt,
            "parameters": parameters
        }
        result = self._query(payload)

        try:
            generated_text = result[0]["generated_text"]
        except (IndexError, KeyError, TypeError):
            generated_text = str(result)
        generated_text = generated_text.strip().split("[/INST]")[-1]

        generated_text = generated_text.strip()
        match = re.search(r'\d+', generated_text)
        if match:
            score = int(match.group(0))
            score = max(0, min(100, score))
        else:
            score = 0
        return score

# Example Usage
# https://api-inference.huggingface.co/models/mistralai/Mistral-7B-
Instruct-v0.2
# https://api-inference.huggingface.co/models/mistralai/Mistral-7B-
Instruct-v0.3
# https://api-inference.huggingface.co/models/mistralai/Mixtral-8x7B-
Instruct-v0.1
# https://api-inference.huggingface.co/models/mistralai/Mistral-Nemo-
Instruct-2407
mistral =
Mistral(api_url="https://api-inference.huggingface.co/models/mistralai
/Mistral-Nemo-Instruct-2407")

concepts = """
1) Proper class definition with __init__ method.
2) Use of encapsulation (private attributes).
3) Correct method definitions.
4) Use of classmethod.
"""

st_code = """
```

```
class Animal:
    \"\"\"This class represents a general animal.\"\"\"

    def __init__(self, name, sound):
        \"\"\"Initializes an Animal object with name and sound.\"\"\"
        self.__name = name
        self.__sound = sound

    def get_name(self):
        \"\"\"Returns the animal's name.\"\"\"
        return self.__name

    def make_sound(self):
        \"\"\"Makes the animal sound.\"\"\"
        print(f"{self.__name} says: {self.__sound}")

    @classmethod
    def create_from_name(cls, name):
        \"\"\"Creates an Animal object from a name.\"\"\"
        return cls(name, "Generic Sound")
"""

print("Check Concepts Output:")
print(mistral.check_concepts(concepts, st_code))

pr_code = """
class Animal:
    def __init__(self, name, sound):
        self.name = name
        self.sound = sound
    def make_sound(self):
        print(f"{self.name} makes a sound: {self.sound}")
"""

score = mistral.compare_code(pr_code, st_code)
print("\nComparison Score:")
print(score)
```

```
Check Concepts Output:
1) yes
2) yes
3) yes
4) yes

Comparison Score:
50
```

Llama Model

```
from huggingface_hub import InferenceClient
```

```python
# Prompt Templates for Concept Checking
SYS_CHECK_PROMPT = """
You are a Python expert and debugging assistant.
Your task is to evaluate the provided Python code against a list of
concepts.
For each concept, verify both its syntactic and logical correctness.
Output exactly one line per concept in the following format:
"<concept number>) yes" or "<concept number>) no"
Do not include any additional commentary, explanations, or error
details.
Ensure the number of output lines matches the number of concepts
provided, with no extra spaces or characters.

Evaluate whether the student's Python code satisfies each of the
following concepts:

{concepts}

Student's Code:

{code}

Provide your evaluation using the exact format specified above.
"""

# Prompt Templates for Code Comparison
SYS_COMPARE_PROMPT = """
You are an AI writing assistant with deep expertise in Python
programming and code debugging.
Carefully analyze and strictly evaluate the Python codes provided by
the user and compare them.
Your output should be a single numerical score representing the
similarity between the two codes.
Do not include any text or formatting besides the score, which should
be between 0 and 100.
Do NOT repeat the system or user prompts in your final response.
"""

COMPARE_PROMPT = """
help me with a score between 0 and 100 that reflects how similar the
professor's code is to the student's code:

Professor's code:
{pr_code}

Student's code:
{st_code}

Instructions:
1. Return just a number between 0 and 100.
```

```
"""

# Llama Class Definition
class Llama:
    def __init__(self, model, api_key):
        self.client = InferenceClient(api_key=api_key)
        self.model = model

    def check_concepts(self, concepts, code, max_tokens=500,
stream=False):
        prompt = SYS_CHECK_PROMPT.format(concepts=concepts, code=code)
        messages = [
            {"role": "user", "content": prompt},
        ]
        response = self.client.chat.completions.create(
            model=self.model,
            messages=messages,
            max_tokens=max_tokens,
            stream=stream
        )
        try:
            return response["choices"][0]["message"]
["content"].strip()
        except (KeyError, TypeError):
            return str(response).strip()

    def compare_code(self, pr_code, st_code, max_tokens=200,
stream=False):
        prompt = SYS_COMPARE_PROMPT.strip() + "\n\n" +
COMPARE_PROMPT.format(pr_code=pr_code, st_code=st_code)
        messages = [
            {"role": "user", "content": prompt},
        ]
        response = self.client.chat.completions.create(
            model=self.model,
            messages=messages,
            max_tokens=max_tokens,
            stream=stream
        )
        try:
            result = response["choices"][0]["message"]
["content"].strip()
        except (KeyError, TypeError):
            result = str(response).strip()

        try:
            score = int(result)
            score = max(0, min(100, score))
        except ValueError:
            match = re.search(r'\d+', result)
```

```python
            if match:
                score = int(match.group(0))
                score = max(0, min(100, score))
            else:
                score = 0

        return score

# Example Usage
model_name = "meta-llama/Llama-3.2-1B-Instruct"
# model_name = "meta-llama/Meta-Llama-3-8B-Instruct"

llama = Llama(model=model_name, api_key=HF_TOKEN)

concepts = """
1) Proper class definition with __init__ method.
2) Use of encapsulation (private attributes).
3) Correct method definitions.
4) Use of classmethod.
"""
student_code = """
class Animal:
    \"\"\"This class represents a general animal.\"\"\"

    def __init__(self, name, sound):
        \"\"\"Initializes an Animal object with name and sound.\"\"\"
        self.__name = name
        self.__sound = sound

    def get_name(self):
        \"\"\"Returns the animal's name.\"\"\"
        return self.__name

    def make_sound(self):
        \"\"\"Makes the animal sound.\"\"\"
        print(f"{self.__name} says: {self.__sound}")

    @classmethod
    def create_from_name(cls, name):
        \"\"\"Creates an Animal object from a name.\"\"\"
        return cls(name, "Generic Sound")
"""
print("Check Concepts Output:")
print(llama.check_concepts(concepts, student_code))

professor_code = """
class Animal:
    def __init__(self, name, sound):
        self.name = name
        self.sound = sound
```

```
    def make_sound(self):
        print(f"{self.name} makes a sound: {self.sound}")
"""

score = llama.compare_code(professor_code, student_code)
print("\nComparison Score:")
print(score)
```

```
Check Concepts Output:
1) yes
2) no
3) no
4) no

Comparison Score:
40
```

Helpers For Score Evaluation

```
def rgb2hex(rgb):
    """Converts an (R, G, B, A) tuple (values between 0 and 1) to a
HEX string."""
    return '#{:02x}{:02x}{:02x}'.format(int(rgb[0]*255),
int(rgb[1]*255), int(rgb[2]*255))

def get_cell_color(metric_name, value):
    lower_is_better_keywords = [
        "error", "loss", "mae", "mse", "rmse", "mape", "rae", "rse",
"quantile", "median"
    ]
    lower_is_better = any(kw in metric_name.lower() for kw in
lower_is_better_keywords)

    # If the value is between 0 and 1, assume it’s a percentage (e.g.,
0.85 -> 85%)
    if 0 <= value <= 1:
        perc = value * 100
        display_str = f"{perc:.2f}%"
    else:
        perc = value
        display_str = f"{value:.2f}"

    discrete_val = round(perc / 10) * 10
    norm = discrete_val / 100.0
    cmap = plt.get_cmap('RdYlGn')

    # For some metrics (loss, error, etc.) lower is better
    if not lower_is_better:
        color = cmap(norm)
    else:
```

```
        color = cmap(1 - norm)

    color_hex = rgb2hex(color)

    return color_hex, display_str


def evaluate_generated_feedback(true_feedbacks, pred_feedbacks,
quantile=0.9):
    if len(true_feedbacks) != len(pred_feedbacks):
        raise ValueError("The number of true feedbacks must match the
number of predicted feedbacks.")

    # Initialize lists to collect per-sample metrics
    exact_matches = 0
    bleu_scores = []
    meteor_scores = []
    rouge1_f1_scores = []
    rouge2_f1_scores = []
    rougel_f1_scores = []
    levenshtein_scores = []

    # Create a ROUGE scorer (using stemming)
    rouge_scorer_obj = rouge_scorer.RougeScorer(['rouge1', 'rouge2',
'rougeL'], use_stemmer=True)
    smoothing_function = SmoothingFunction().method1

    # Loop over each pair of true and predicted feedback
    for true_fb, pred_fb in zip(true_feedbacks, pred_feedbacks):
        # Check for an exact (after stripping) match:
        if true_fb.strip() == pred_fb.strip():
            exact_matches += 1

        # Tokenize the strings (using a simple whitespace split here;
you might use nltk.word_tokenize for more sophisticated tokenization)
        true_tokens = true_fb.split()
        pred_tokens = pred_fb.split()

        # --- BLEU Score ---
        # Note: sentence_bleu expects a list of reference token lists.
        bleu = sentence_bleu([true_tokens], pred_tokens,
smoothing_function=smoothing_function)
        bleu_scores.append(bleu)

        # --- METEOR Score ---
        try:
            m_score = meteor_score([true_fb], pred_fb)
        except Exception:
            m_score = 0.0
        meteor_scores.append(m_score)
```

```
        # --- ROUGE Scores ---
        scores = rouge_scorer_obj.score(true_fb, pred_fb)
        rouge1_f1_scores.append(scores['rouge1'].fmeasure)
        rouge2_f1_scores.append(scores['rouge2'].fmeasure)
        rougel_f1_scores.append(scores['rougeL'].fmeasure)

        # --- Levenshtein Similarity ---
        # Using difflib’s SequenceMatcher ratio as a proxy for text
similarity.
        sim_ratio = difflib.SequenceMatcher(None, true_fb,
pred_fb).ratio()
        levenshtein_scores.append(sim_ratio)

    # Aggregate the metrics across the entire dataset
    exact_match_accuracy = exact_matches / len(true_feedbacks)
    avg_bleu = np.mean(bleu_scores)
    avg_meteor = np.mean(meteor_scores)
    avg_rouge1 = np.mean(rouge1_f1_scores)
    avg_rouge2 = np.mean(rouge2_f1_scores)
    avg_rougel = np.mean(rougel_f1_scores)
    avg_levenshtein = np.mean(levenshtein_scores)

    metrics_dict = {
        "Exact Match Accuracy": exact_match_accuracy,
        "Average BLEU Score": avg_bleu,
        "Average METEOR Score": avg_meteor,
        "Average ROUGE-1 F1": avg_rouge1,
        "Average ROUGE-2 F1": avg_rouge2,
        "Average ROUGE-L F1": avg_rougel,
        "Average Levenshtein Similarity": avg_levenshtein
    }

    # --- BERTScore ---
    try:
        P, R, F1 = bertscore(pred_feedbacks, true_feedbacks,
lang='en', verbose=True)
        avg_bert_f1 = F1.mean().item()
        metrics_dict["Average BERTScore F1"] = avg_bert_f1
    except Exception as e:
        print("Error computing BERTScore:", e)


    # --- Create a DataFrame & visualize metrics in a table ---
    metrics_df = pd.DataFrame(list(metrics_dict.items()),
columns=["Metric", "Value"])

    fig_table, ax_table = plt.subplots(figsize=(6, len(metrics_df) *
0.3 + 1))
    ax_table.axis('off')
```

```
    table = ax_table.table(
        cellText=metrics_df.values,
        colLabels=metrics_df.columns,
        cellLoc='center',
        loc='center'
    )
    table.auto_set_font_size(False)
    table.set_fontsize(10)
    table.scale(1, 1.5)

    # Color the metric values based on the metric (using
get_cell_color)
    for (row, col), cell in table.get_celld().items():
        if row == 0:
            cell.set_facecolor("#f0f0f0")
            continue
        if col == 1:
            metric_name = metrics_df.iloc[row - 1]["Metric"]
            metric_value = metrics_df.iloc[row - 1]["Value"]
            color, display_str = get_cell_color(metric_name,
metric_value)
            cell.get_text().set_text(display_str)
            cell.set_facecolor(color)

    plt.title("Generated Feedback Evaluation Metrics", fontsize=14,
pad=20)
    plt.tight_layout()
    plt.show()

    # --- Additional Visualizations ---
    fig, axs = plt.subplots(2, 3, figsize=(18, 10))
    axs = axs.flatten()

    axs[0].hist(bleu_scores, bins=20, edgecolor='k', alpha=0.7)
    axs[0].set_title("BLEU Score Distribution")
    axs[1].hist(meteor_scores, bins=20, edgecolor='k', alpha=0.7)
    axs[1].set_title("METEOR Score Distribution")
    axs[2].hist(rouge1_f1_scores, bins=20, edgecolor='k', alpha=0.7)
    axs[2].set_title("ROUGE-1 F1 Distribution")
    axs[3].hist(rouge2_f1_scores, bins=20, edgecolor='k', alpha=0.7)
    axs[3].set_title("ROUGE-2 F1 Distribution")
    axs[4].hist(rougel_f1_scores, bins=20, edgecolor='k', alpha=0.7)
    axs[4].set_title("ROUGE-L F1 Distribution")
    axs[5].hist(levenshtein_scores, bins=20, edgecolor='k', alpha=0.7)
    axs[5].set_title("Levenshtein Similarity Distribution")

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

```
    return metrics_dict

def evaluate_check_concepts(model, dataset):
    tr_feedbacks = []
    pr_feedbacks = []

    for idx, entry in tqdm(enumerate(dataset), total=len(dataset),
desc="Processing Inference"):
        # get our variables
        concepts = entry.get("concepts")
        st_code = entry.get("st_code")
        tr_fb = entry.get("feedback")

        # raise an error if we don't have all the variables
        if not concepts or not st_code or not tr_fb:
            raise ValueError("Missing required fields in the
dataset.")

        # store tr_scores for eval later :)
        tr_feedbacks.append(tr_fb)

        # this could break the evaluation
        pr_fb = model.check_concepts(
            concepts=concepts,
            code=st_code,
        )

        # predicted score
        pr_feedbacks.append(pr_fb)

        # tqdm.write(f"\nProgress: {idx + 1}/{len(dataset)}")
        # tqdm.write(f"Predicted Feedback: \n{pr_fb}")
        # tqdm.write(f"True Feedback     : \n{tr_fb}\n")

        if idx == 20:
            break

    # now let's evaluate
    _ = evaluate_generated_feedback(tr_feedbacks, pr_feedbacks,
quantile=0.9)
```

Load Evaluation Data

```
data_path = "/content/check_concept_dataset_1.json"

with open(data_path, "r") as f:
    dataset = json.load(f)

print(dataset[0]["st_code"])
```

```python
print(dataset[0]["concepts"])
print(dataset[0]["feedback"])
```

```
def create_person(person_name, person_age):
    person = {"person_name": person_name, "person_age": person_age}
    return person

1. Concept of a class.
2. Concept of private variables inside the class.
3. Concept of class attributes.
4. Concept of types in Python.
1) No
2) No
3) No
4) No
```

Evaluate Smol Model

Mistral

```python
API_URL =
"https://api-inference.huggingface.co/models/mistralai/Mistral-7B-
Instruct-v0.2"
mistral = Mistral(api_url=API_URL)
evaluate_check_concepts(
    model=mistral,
    dataset=dataset,
)
```

```
Processing Inference:   8%|█          | 20/248 [00:23<04:31,  1.19s/it]
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py
:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
access public models or datasets.
  warnings.warn(
```


{"model_id":"c8b7cc5e61fd40f9aa6fe66e18d3de9c","version_major":2,"version_minor":0}

{"model_id":"bb7607647fce4058997e244d17cbb985","version_major":2,"version_minor":0}

{"model_id":"91a0ea3b07184b22a7aa55fd2b18fc1d","version_major":2,"version_minor":0}

{"model_id":"07be9518f1c3413099d1e4b4e75d36f7","version_major":2,"version_minor":0}

{"model_id":"0701440b0e75440c8ca61991aa8e159e","version_major":2,"version_minor":0}

{"model_id":"1de98bcc0fd24d9c978200bb8e4174e6","version_major":2,"version_minor":0}

```
Some weights of RobertaModel were not initialized from the model
checkpoint at roberta-large and are newly initialized:
['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
```

```
calculating scores...
computing bert embedding.
```

{"model_id":"399145ae7db14974825f22c42deb71b7","version_major":2,"version_minor":0}

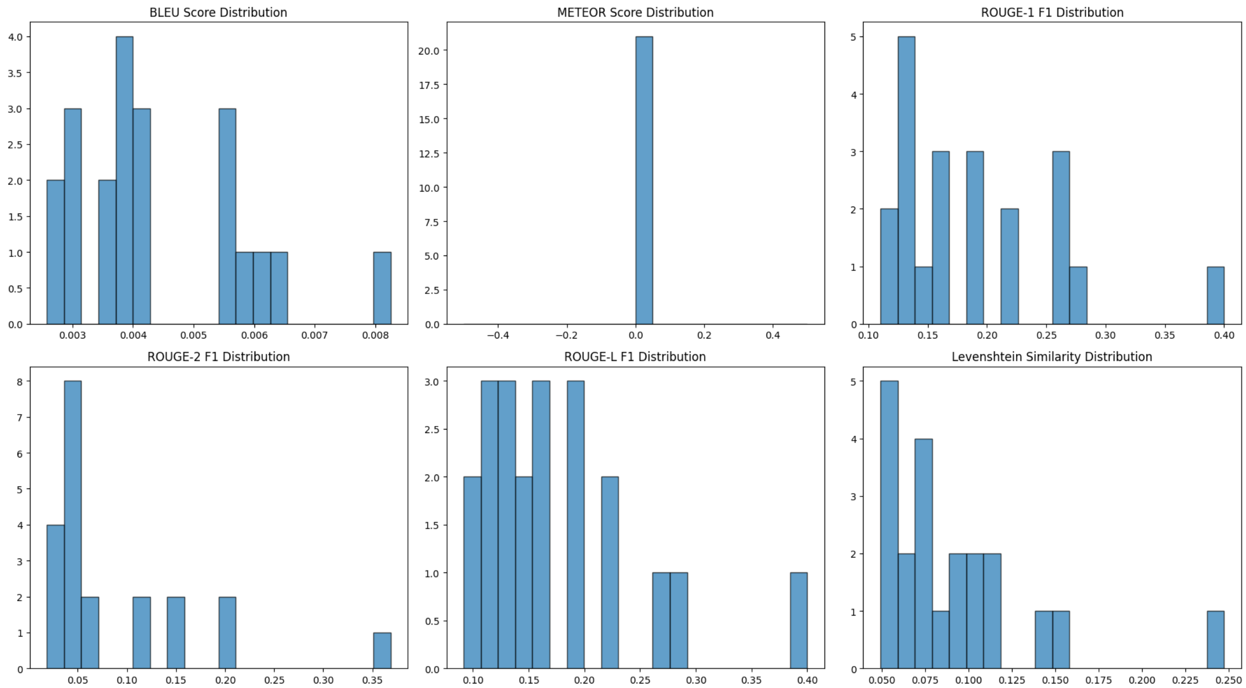
```
computing greedy matching.
```

{"model_id":"329fbe898fc84903aa18e171c09a170b","version_major":2,"version_minor":0}

```
done in 30.25 seconds, 0.69 sentences/sec
```

Generated Feedback Evaluation Metrics

Metric	Value
Exact Match Accuracy	0.00%
Average BLEU Score	0.44%
Average METEOR Score	0.00%
Average ROUGE-1 F1	19.12%
Average ROUGE-2 F1	8.79%
Average ROUGE-L F1	17.49%
Average Levenshtein Similarity	9.24%
Average BERTScore F1	85.26%




```python
API_URL =
"https://api-inference.huggingface.co/models/mistralai/Mistral-7B-Instruct-v0.3"
mistral = Mistral(api_url=API_URL)
evaluate_check_concepts(
    model=mistral,
    dataset=dataset,
)
```

```
Processing Inference:   8%|█         | 20/248 [00:15<03:02,  1.25it/s]
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

calculating scores...
computing bert embedding.
```

{"model_id":"0baeb80dbd2041de9a30af529fd074ea","version_major":2,"version_minor":0}

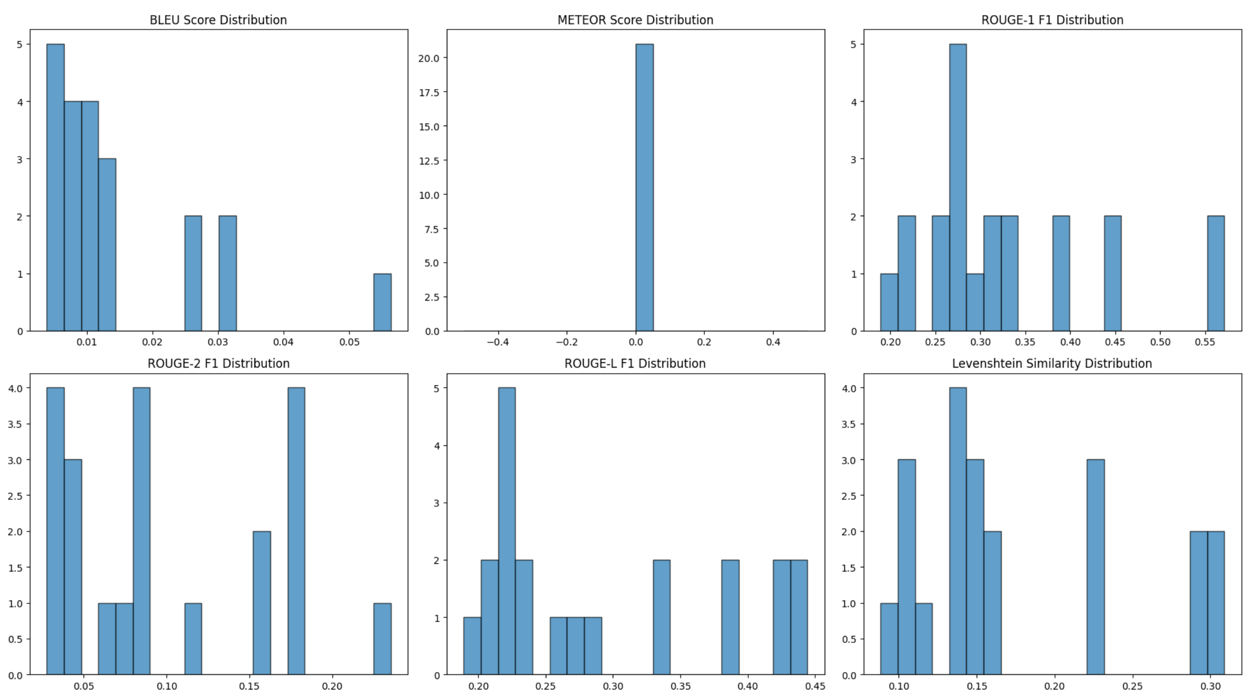
```
computing greedy matching.
```

{"model_id":"6d9173d7ef504dd3aa622c89012ddf29","version_major":2,"version_minor":0}

```
done in 18.79 seconds, 1.12 sentences/sec
```

Generated Feedback Evaluation Metrics

Metric	Value
Exact Match Accuracy	0.00%
Average BLEU Score	1.46%
Average METEOR Score	0.00%
Average ROUGE-1 F1	32.97%
Average ROUGE-2 F1	10.03%
Average ROUGE-L F1	29.53%
Average Levenshtein Similarity	17.69%
Average BERTScore F1	85.75%




```
API_URL =
"https://api-inference.huggingface.co/models/mistralai/Mixtral-8x7B-
Instruct-v0.1"
mistral = Mistral(api_url=API_URL)
evaluate_check_concepts(
    model=mistral,
    dataset=dataset,
)
```

```
Processing Inference:   8%|█          | 20/248 [00:26<04:56,  1.30s/it]
Some weights of RobertaModel were not initialized from the model
checkpoint at roberta-large and are newly initialized:
['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.

calculating scores...
computing bert embedding.
```

{"model_id":"92e109d0dba04c4bbe2fb454563eafe4","version_major":2,"version_minor":0}

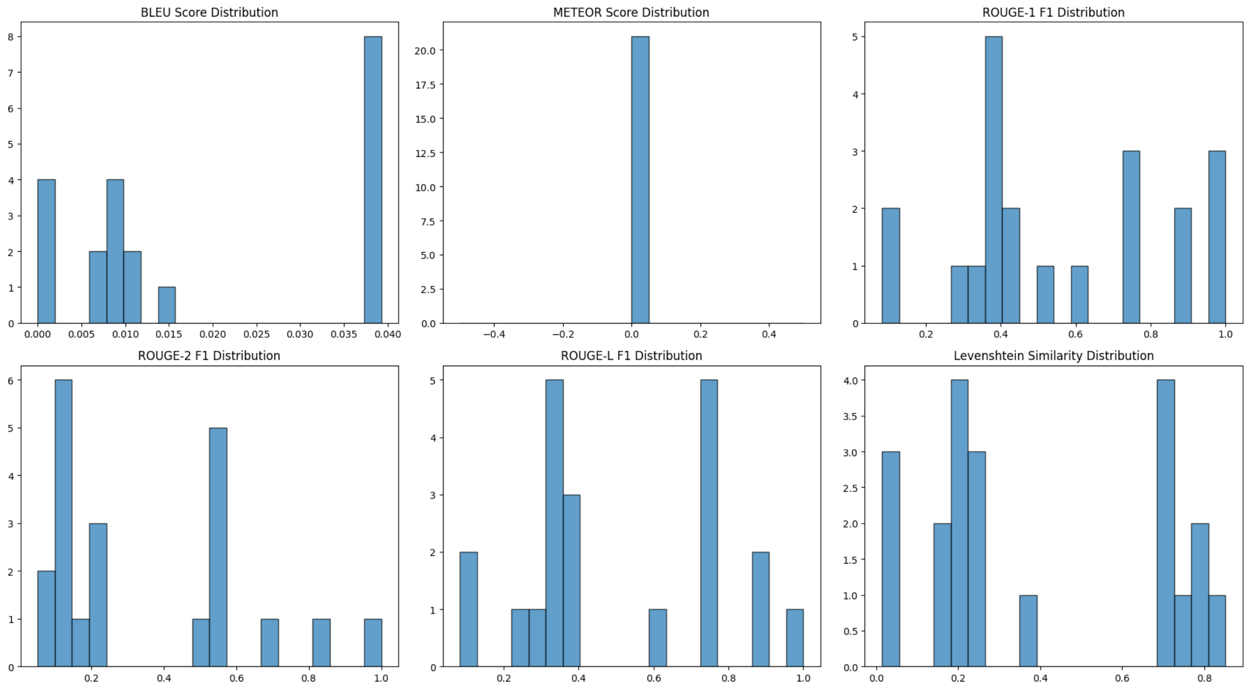
```
computing greedy matching.
```

{"model_id":"924970f3e85c47b196b13cdbf83945ec","version_major":2,"version_minor":0}

```
done in 52.92 seconds, 0.40 sentences/sec
```

Generated Feedback Evaluation Metrics

Metric	Value
Exact Match Accuracy	0.00%
Average BLEU Score	1.91%
Average METEOR Score	0.00%
Average ROUGE-1 F1	55.39%
Average ROUGE-2 F1	36.38%
Average ROUGE-L F1	50.25%
Average Levenshtein Similarity	39.23%
Average BERTScore F1	91.13%




```
API_URL =
"https://api-inference.huggingface.co/models/mistralai/Mistral-Nemo-
Instruct-2407"
mistral = Mistral(api_url=API_URL)
evaluate_check_concepts(
    model=mistral,
    dataset=dataset,
)
```

```
Processing Inference:   8%|█         | 20/248 [00:11<02:08,  1.77it/s]
Some weights of RobertaModel were not initialized from the model
checkpoint at roberta-large and are newly initialized:
['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.

calculating scores...
computing bert embedding.
```


{"model_id":"80ef1320bcd540b0a25c12cde12c33b9","version_major":2,"vers
ion_minor":0}

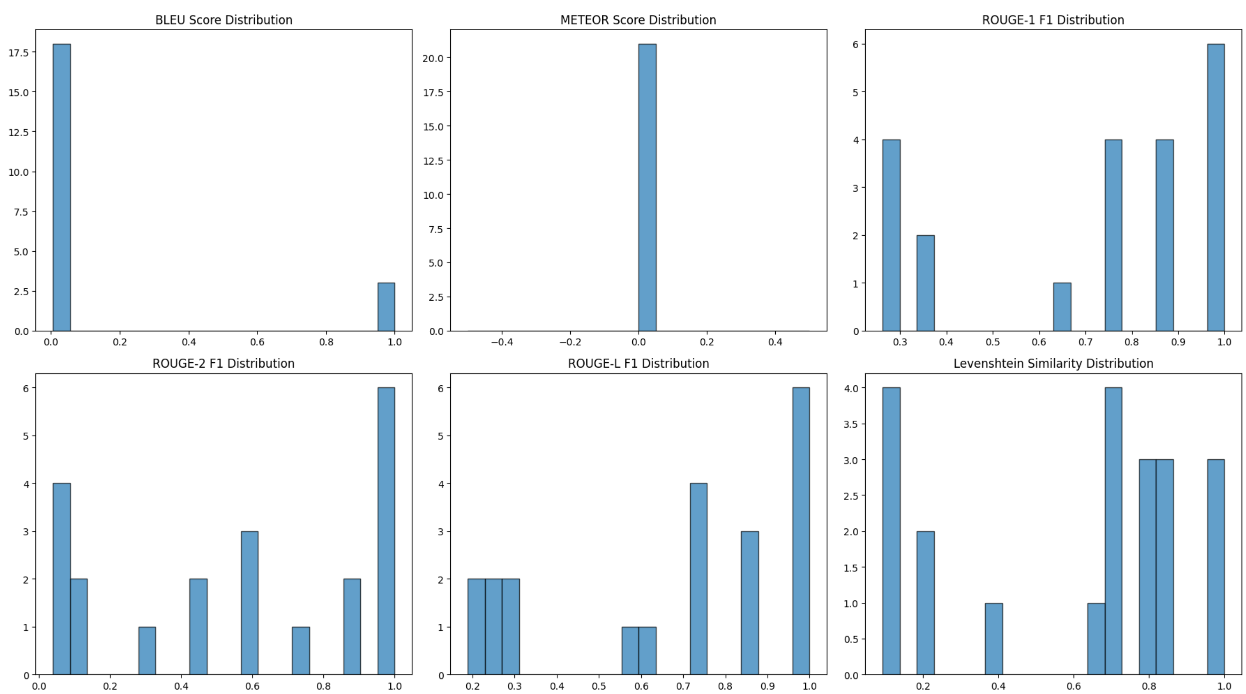

```
computing greedy matching.
```


{"model_id":"09b6fbb63312425aadcc0f87c180a0d5","version_major":2,"vers
ion_minor":0}


```
done in 19.43 seconds, 1.08 sentences/sec
```

Generated Feedback Evaluation Metrics

Metric	Value
Exact Match Accuracy	14.29%
Average BLEU Score	16.63%
Average METEOR Score	0.00%
Average ROUGE-1 F1	71.05%
Average ROUGE-2 F1	55.73%
Average ROUGE-L F1	67.87%
Average Levenshtein Similarity	60.21%
Average BERTScore F1	94.58%



Llama

```
model_name = "meta-llama/Llama-3.2-1B-Instruct"
llama = Llama(model=model_name, api_key=HF_TOKEN)
evaluate_check_concepts(
    model=llama,
    dataset=dataset,
)
```

```
Processing Inference:   8%|█          | 20/248 [00:03<00:43,  5.26it/s]
Some weights of RobertaModel were not initialized from the model
checkpoint at roberta-large and are newly initialized:
['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.

calculating scores...
computing bert embedding.
```

{"model_id":"07a586d37d614252b000709c48830056","version_major":2,"version_minor":0}

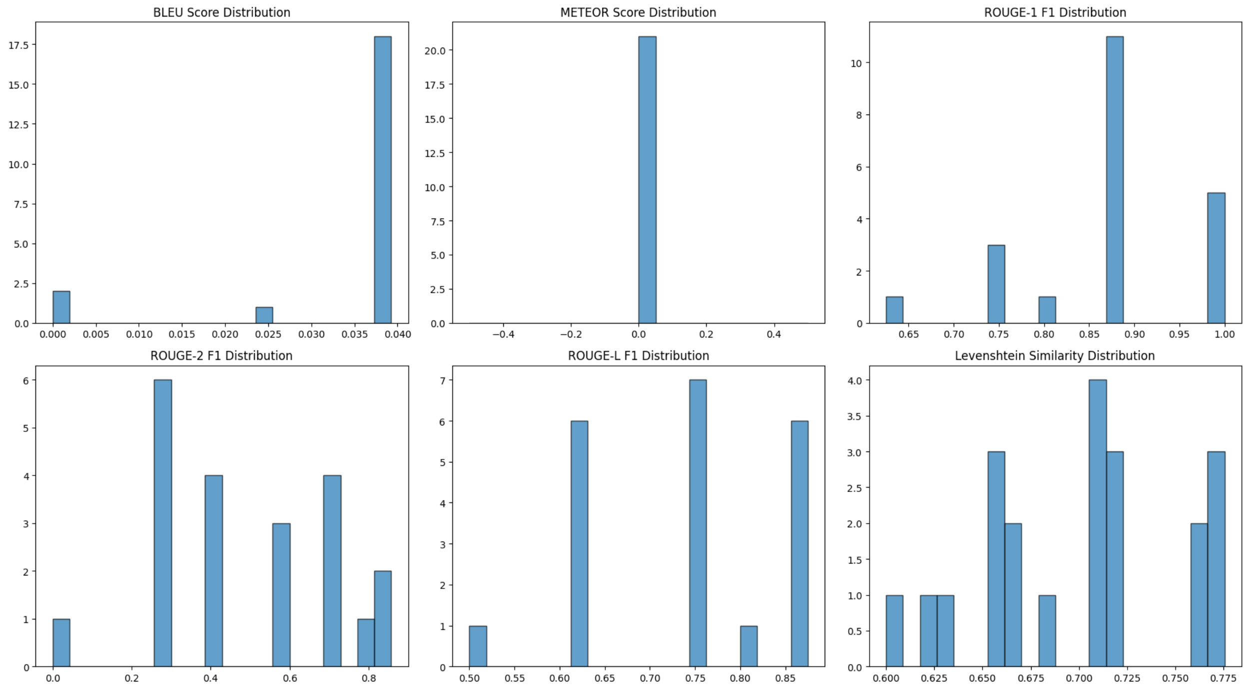
```
computing greedy matching.
```

{"model_id":"a13bfb46392f4bdf93e25d570307a0c2","version_major":2,"version_minor":0}

```
done in 3.71 seconds, 5.66 sentences/sec
```

Generated Feedback Evaluation Metrics

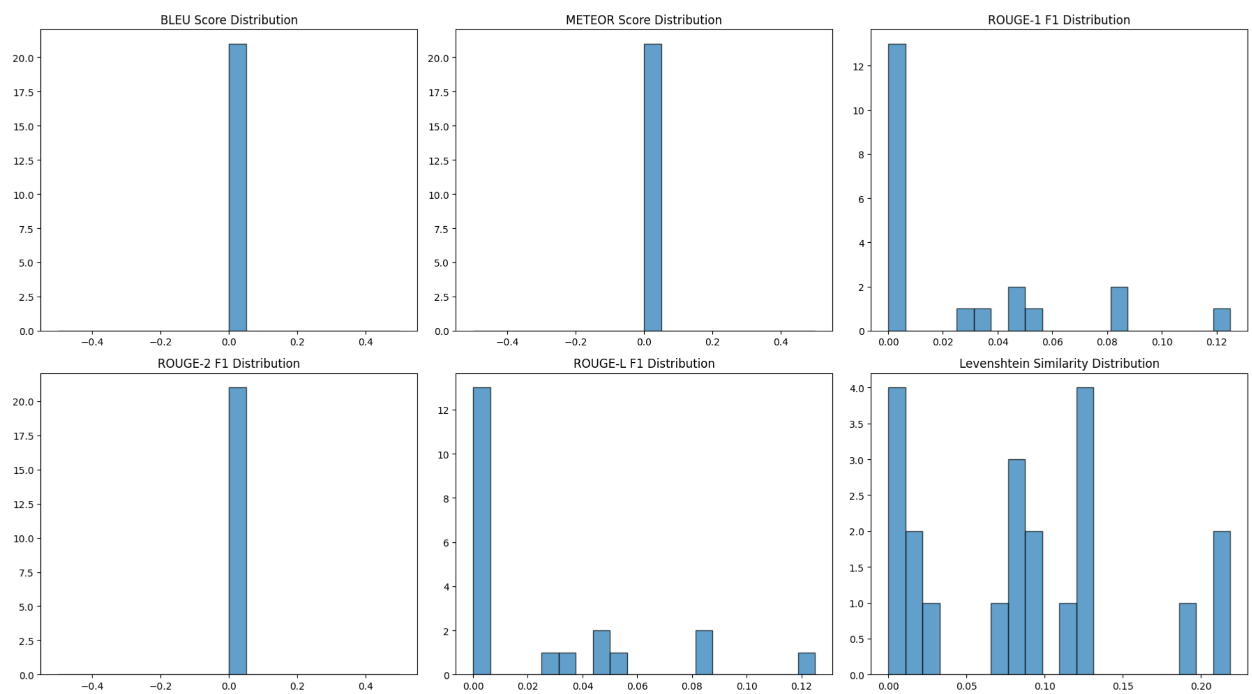
Metric	Value
Exact Match Accuracy	0.00%
Average BLEU Score	3.48%
Average METEOR Score	0.00%
Average ROUGE-1 F1	87.14%
Average ROUGE-2 F1	49.96%
Average ROUGE-L F1	74.05%
Average Levenshtein Similarity	69.85%
Average BERTScore F1	95.63%




```
model_name = "meta-llama/Meta-Llama-3-8B-Instruct"
llama = Llama(model=model_name, api_key=HF_TOKEN)
evaluate_check_concepts(
    model=llama,
    dataset=dataset,
)

Processing Inference:   8%|█          | 20/248 [00:14<02:43,  1.39it/s]
Some weights of RobertaModel were not initialized from the model
checkpoint at roberta-large and are newly initialized:
['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.

calculating scores...
computing bert embedding.

{"model_id":"741cbf56a65b44bba51ae5f0173eace8","version_major":2,"vers
ion_minor":0}

computing greedy matching.

{"model_id":"960e4680d909469c96b080839ef70b04","version_major":2,"vers
ion_minor":0}

done in 37.74 seconds, 0.56 sentences/sec
```

Generated Feedback Evaluation Metrics

Metric	Value
Exact Match Accuracy	0.00%
Average BLEU Score	0.00%
Average METEOR Score	0.00%
Average ROUGE-1 F1	2.43%
Average ROUGE-2 F1	0.00%
Average ROUGE-L F1	2.43%
Average Levenshtein Similarity	8.67%
Average BERTScore F1	77.23%



Gemma

```
gemma = Gemma(model="google/gemma-2-2b-it")
evaluate_check_concepts(
    model=gemma,
    dataset=dataset,
)

Processing Inference:   8%|█          | 20/248 [00:06<01:17,  2.94it/s]
Some weights of RobertaModel were not initialized from the model
```

```
checkpoint at roberta-large and are newly initialized:
['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.

calculating scores...
computing bert embedding.
```

{"model_id":"34a942037a7b46a3a725ac41d949017b","version_major":2,"version_minor":0}

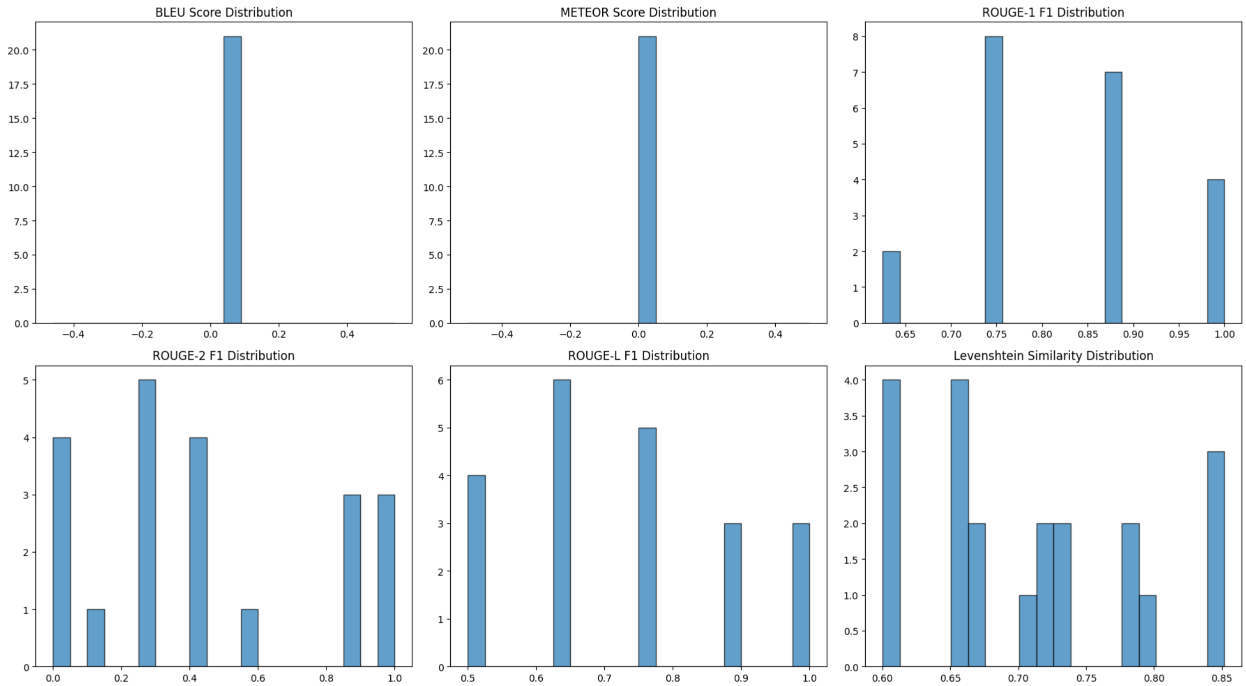
```
computing greedy matching.
```

{"model_id":"57934a27de214d35bef5391d551f45a8","version_major":2,"version_minor":0}

```
done in 3.51 seconds, 5.99 sentences/sec
```

Generated Feedback Evaluation Metrics

Metric	Value
Exact Match Accuracy	0.00%
Average BLEU Score	3.93%
Average METEOR Score	0.00%
Average ROUGE-1 F1	82.74%
Average ROUGE-2 F1	44.90%
Average ROUGE-L F1	72.02%
Average Levenshtein Similarity	70.75%
Average BERTScore F1	97.06%




```
gemma = Gemma(model="google/gemma-2-9b-it")
evaluate_check_concepts(
    model=gemma,
    dataset=dataset,
)
```

```
Processing Inference:   8%|█          | 20/248 [00:13<02:30,  1.52it/s]
Some weights of RobertaModel were not initialized from the model
checkpoint at roberta-large and are newly initialized:
['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.

calculating scores...
computing bert embedding.
```

{"model_id":"ad77b352ca3045ae824a8b318f650f1c","version_major":2,"version_minor":0}

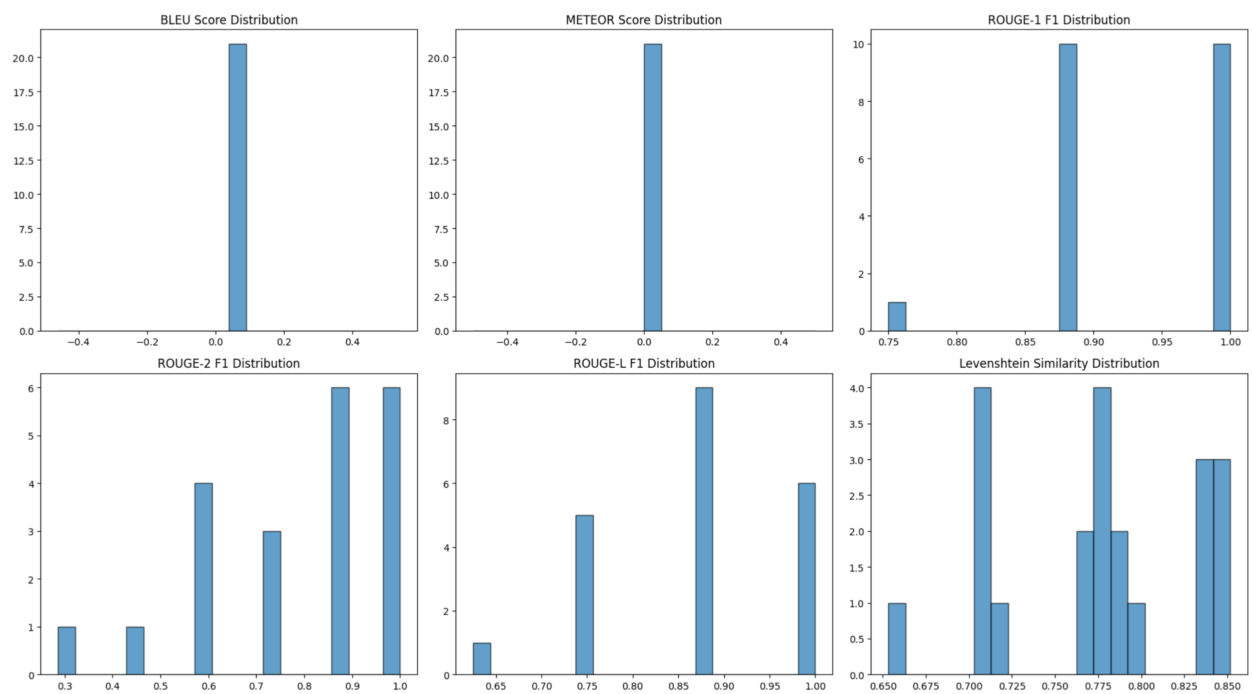
```
computing greedy matching.
```

{"model_id":"489679ad8dee46c9afa3cc44ea12ea57","version_major":2,"version_minor":0}

```
done in 2.84 seconds, 7.38 sentences/sec
```

Generated Feedback Evaluation Metrics

Metric	Value
Exact Match Accuracy	0.00%
Average BLEU Score	3.93%
Average METEOR Score	0.00%
Average ROUGE-1 F1	92.86%
Average ROUGE-2 F1	77.55%
Average ROUGE-L F1	86.90%
Average Levenshtein Similarity	77.39%
Average BERTScore F1	97.59%




```
gemma = Gemma(model="google/gemma-2-27b-it")
evaluate_check_concepts(
    model=gemma,
    dataset=dataset,
)
```

```
Processing Inference:   8%|█          | 20/248 [00:27<05:08,  1.35s/it]
Some weights of RobertaModel were not initialized from the model
checkpoint at roberta-large and are newly initialized:
```

```
['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.

calculating scores...
computing bert embedding.
```

{"model_id":"b871de673bd14ce99223fc6146b53f6a","version_major":2,"version_minor":0}

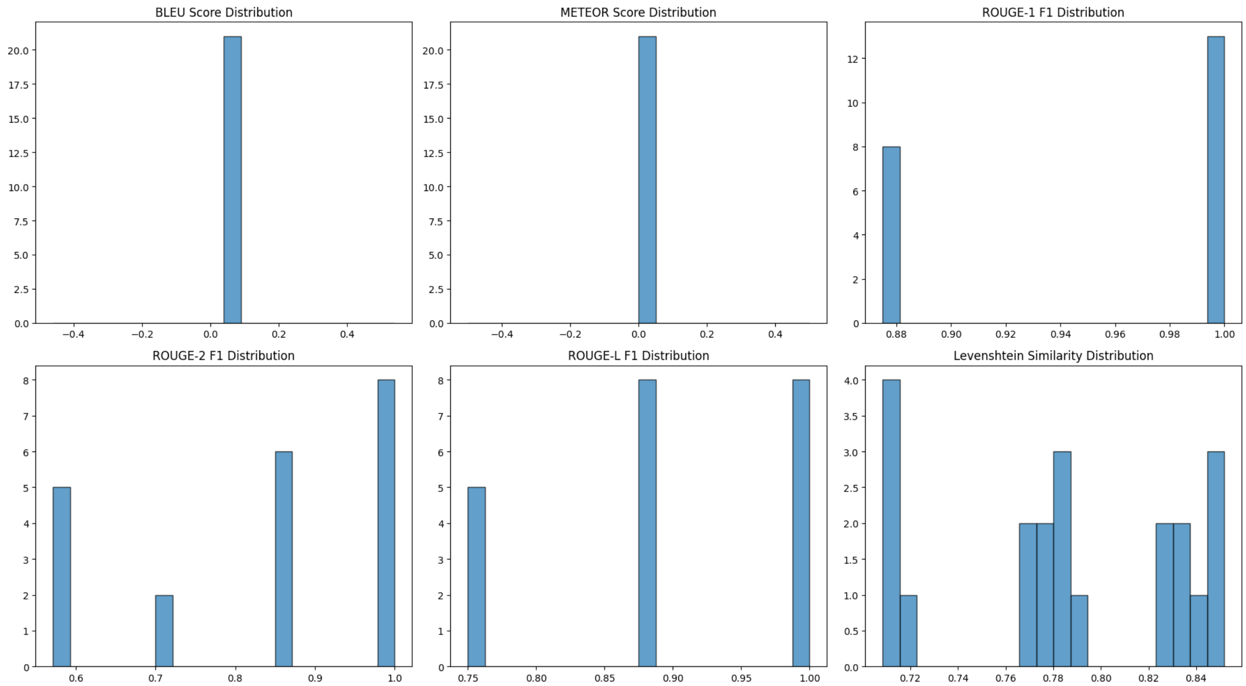
```
computing greedy matching.
```

{"model_id":"f7d51b1685b74e819525e216270a3b81","version_major":2,"version_minor":0}

```
done in 3.02 seconds, 6.95 sentences/sec
```

Generated Feedback Evaluation Metrics

Metric	Value
Exact Match Accuracy	0.00%
Average BLEU Score	3.93%
Average METEOR Score	0.00%
Average ROUGE-1 F1	95.24%
Average ROUGE-2 F1	82.99%
Average ROUGE-L F1	89.29%
Average Levenshtein Similarity	78.47%
Average BERTScore F1	97.56%



Smol

```
smol_lm = SmolLM(checkpoint="HuggingFaceTB/SmolLM2-1.7B-Instruct")

evaluate_check_concepts(
    model=smol_lm,
    dataset=dataset,
)

Processing Inference:   0%|          | 0/248 [00:00<?, ?it/s]The attention mask is not set and cannot be inferred from input because pad token is same as eos token. As a consequence, you may observe unexpected behavior. Please pass your input's `attention_mask` to obtain reliable results.
Processing Inference:   8%|█         | 20/248 [15:36<2:57:58, 46.84s/it]
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

calculating scores...
computing bert embedding.
```

{"model_id":"0919b91a436c43e18e1567a16231c366","version_major":2,"version_minor":0}

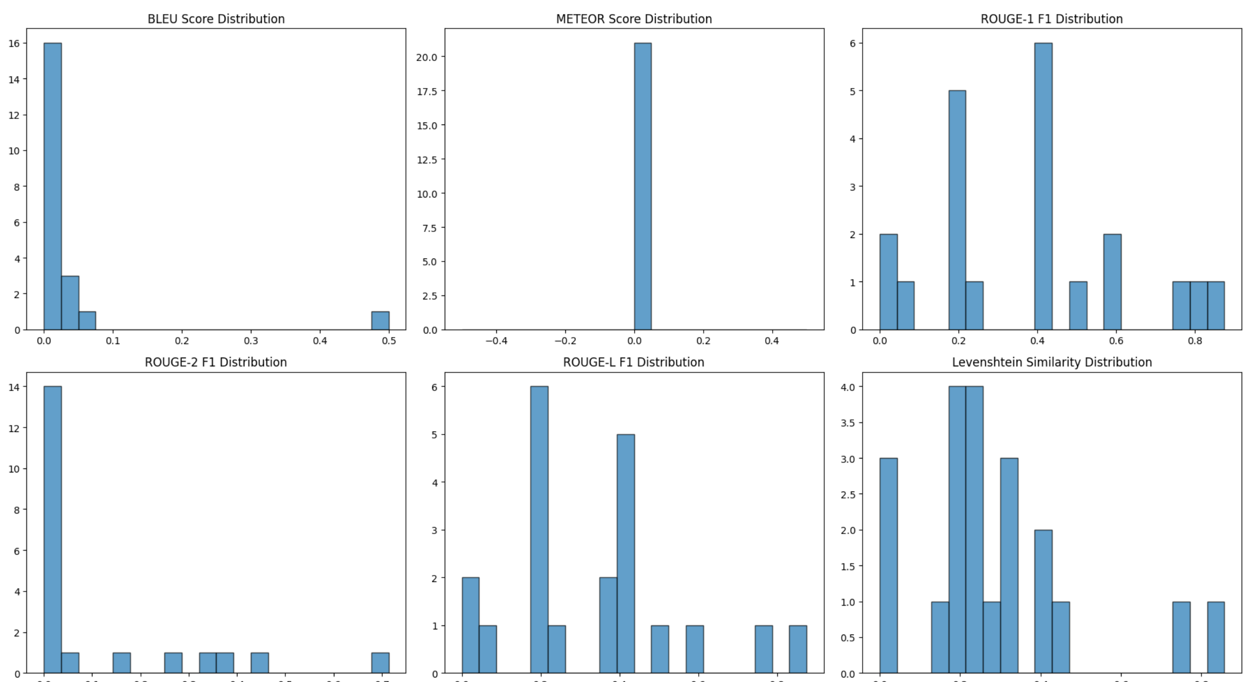
```
computing greedy matching.
```

{"model_id":"df5bd70d43fc4b788fc171cd38c8bc5f","version_major":2,"version_minor":0}

```
done in 110.11 seconds, 0.19 sentences/sec
```

Generated Feedback Evaluation Metrics

Metric	Value
Exact Match Accuracy	0.00%
Average BLEU Score	3.41%
Average METEOR Score	0.00%
Average ROUGE-1 F1	37.27%
Average ROUGE-2 F1	11.04%
Average ROUGE-L F1	33.41%
Average Levenshtein Similarity	28.81%
Average BERTScore F1	84.29%




```
smol_lm = SmolLM(checkpoint="HuggingFaceTB/SmolLM2-135M-Instruct")

evaluate_check_concepts(
    model=smol_lm,
```

```
    dataset=dataset,
)

Processing Inference:   8%|█          | 20/248 [17:45<3:22:28,
53.28s/it]
Some weights of RobertaModel were not initialized from the model
checkpoint at roberta-large and are newly initialized:
['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.

calculating scores...
computing bert embedding.
```

{"model_id":"0102228996f4419fae38a9aaf71d8b9e","version_major":2,"version_minor":0}

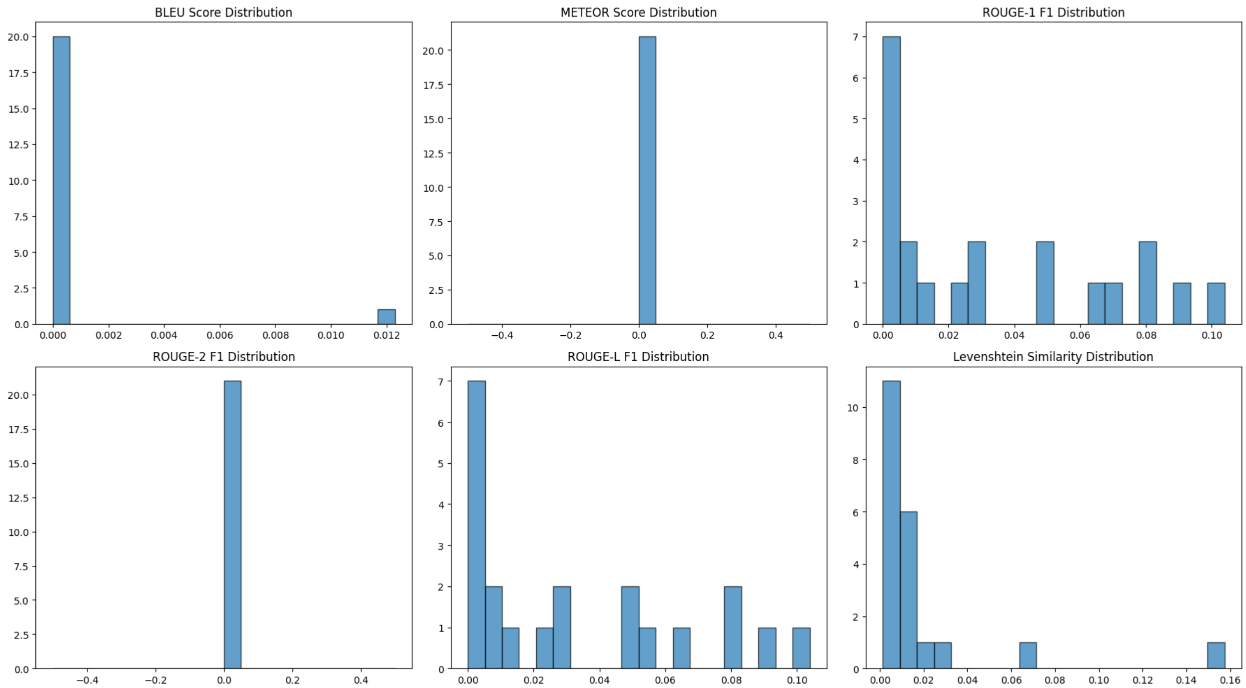
```
computing greedy matching.
```

{"model_id":"7ccaa6c6088f4881b660b19aa70ad5bb","version_major":2,"version_minor":0}

```
done in 145.24 seconds, 0.14 sentences/sec
```

Generated Feedback Evaluation Metrics

Metric	Value
Exact Match Accuracy	0.00%
Average BLEU Score	0.06%
Average METEOR Score	0.00%
Average ROUGE-1 F1	3.42%
Average ROUGE-2 F1	0.00%
Average ROUGE-L F1	3.35%
Average Levenshtein Similarity	1.96%
Average BERTScore F1	75.31%




```python
smol_lm = SmolLM(checkpoint="HuggingFaceTB/SmolLM2-360M-Instruct")

evaluate_check_concepts(
    model=smol_lm,
    dataset=dataset,
)
```

```
Processing Inference:   8%|█          | 20/248 [05:33<1:03:24,
16.69s/it]
Some weights of RobertaModel were not initialized from the model
checkpoint at roberta-large and are newly initialized:
['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.

calculating scores...
computing bert embedding.
```

{"model_id":"6e60652ced0242c6a521974f383550e4","version_major":2,"version_minor":0}

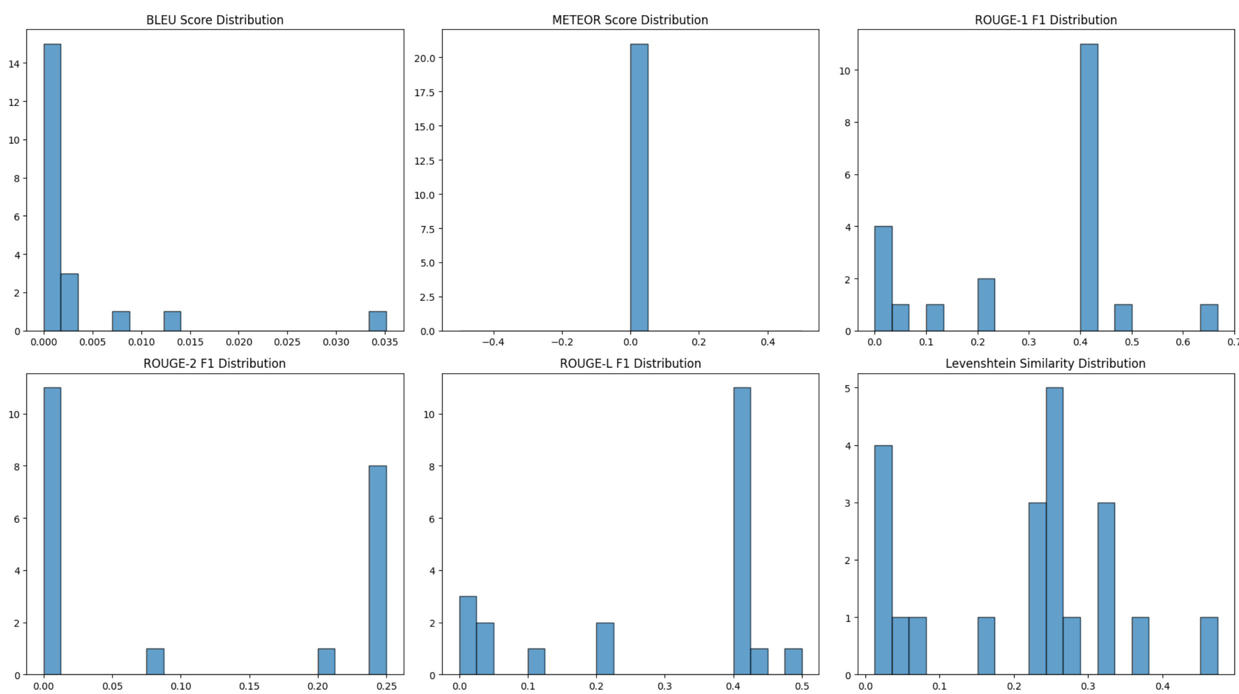
```
computing greedy matching.
```

{"model_id":"f08e469af618442d83e3e665e0e7c7fc","version_major":2,"version_minor":0}

```
done in 112.35 seconds, 0.19 sentences/sec
```

Generated Feedback Evaluation Metrics

Metric	Value
Exact Match Accuracy	0.00%
Average BLEU Score	0.31%
Average METEOR Score	0.00%
Average ROUGE-1 F1	29.44%
Average ROUGE-2 F1	10.93%
Average ROUGE-L F1	28.31%
Average Levenshtein Similarity	21.08%
Average BERTScore F1	82.07%




```
smol_lm = SmolLM(checkpoint="HuggingFaceTB/SmolLM-1.7B-Instruct")

evaluate_check_concepts(
    model=smol_lm,
    dataset=dataset,
)

/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/
_auth.py:94: UserWarning:
```

```
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
access public models or datasets.
  warnings.warn(
```

{"model_id":"c4f764c99f23438c9ac05ed4bdcc2200","version_major":2,"version_minor":0}

{"model_id":"39260905eeda4442bebdd813e296ae29","version_major":2,"version_minor":0}

{"model_id":"62e41368a1e041cfbbe0f0ce6fff3041","version_major":2,"version_minor":0}

{"model_id":"baaa0541847f4b67a1579a0af0f347ae","version_major":2,"version_minor":0}

{"model_id":"4a1cb5d2c84a449cbf8b126cd59e9908","version_major":2,"version_minor":0}

{"model_id":"5e2e1de52bc64c3996d7a4cda00562eb","version_major":2,"version_minor":0}

{"model_id":"966ee671b1e947b68e040f2835fb7fc7","version_major":2,"version_minor":0}

{"model_id":"f136728238064e4580e95c36eb9bc13a","version_major":2,"version_minor":0}

```
Processing Inference:   0%|          | 0/248 [00:00<?, ?it/s]The
attention mask is not set and cannot be inferred from input because
pad token is same as eos token. As a consequence, you may observe
unexpected behavior. Please pass your input's `attention_mask` to
obtain reliable results.
Processing Inference:   8%|█         | 20/248 [2:03:51<23:31:58,
371.57s/it]
```

{"model_id":"7d70d9f3fcfc4ad8ae1e774d5a359ffb","version_major":2,"version_minor":0}

{"model_id":"9cdf9a3803974fc9821885555497e51a","version_major":2,"version_minor":0}

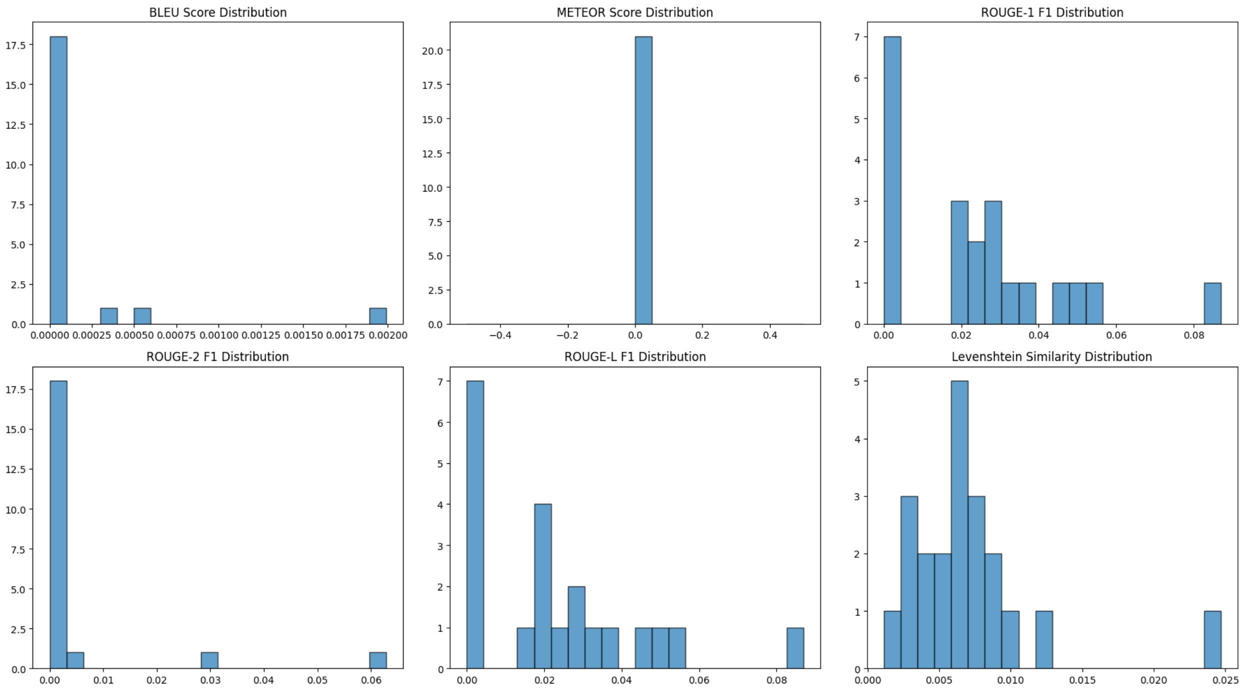
{"model_id":"7344ec8da8c34e2daf92af301d1b5c0c","version_major":2,"version_minor":0}

{"model_id":"9314e97a46c54927a98cbe86e52f7981","version_major":2,"version_minor":0}

```
{"model_id":"b01b8dfd39f6498aba1a3e908af176c5","version_major":2,"version_minor":0}

{"model_id":"85c54ca0888c429386e9539d5cde7e50","version_major":2,"version_minor":0}

Some weights of RobertaModel were not initialized from the model
checkpoint at roberta-large and are newly initialized:
['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.

calculating scores...
computing bert embedding.

{"model_id":"43089cd81e034935a4a53061f43f8742","version_major":2,"version_minor":0}

computing greedy matching.

{"model_id":"b84ba4d731184e58a0546da836bf5e44","version_major":2,"version_minor":0}

done in 193.67 seconds, 0.11 sentences/sec
```

Generated Feedback Evaluation Metrics

Metric	Value
Exact Match Accuracy	0.00%
Average BLEU Score	0.01%
Average METEOR Score	0.00%
Average ROUGE-1 F1	2.37%
Average ROUGE-2 F1	0.46%
Average ROUGE-L F1	2.30%
Average Levenshtein Similarity	0.71%
Average BERTScore F1	74.19%


BLEU Score Distribution
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0.00000
0.00025
0.00050
0.00075
0.00100
0.00125
0.00150
0.00175
0.00200
METEOR Score Distribution
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
−0.4
−0.2
0.0
0.2
0.4
ROUGE-1 F1 Distribution
0
1
2
3
4
5
6
7
0.00
0.02
0.04
0.06
0.08
ROUGE-2 F1 Distribution
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0.00
0.01
0.02
0.03
0.04
0.05
0.06
ROUGE-L F1 Distribution
0
1
2
3
4
5
6
7
0.00
0.02
0.04
0.06
0.08
Levenshtein Similarity Distribution
0
1
2
3
4
5
0.000
0.005
0.010
0.015
0.020
0.025