6_0_2_Model_Evaluation_and_Error_Analysis—Multi

April 13, 2025

0.1 Packages, Library Imports, File Mounts, & Data Imports ** Run All **

[ ]:
```
!pip install -q transformers
!pip install -q torchinfo
!pip install -q datasets
!pip install -q evaluate
!pip install -q nltk
!pip install -q contractions
!pip install -q hf_xet
!pip install -q sentencepiece
# !pip install -q import openpyxl
```

[ ]:
```
!sudo apt-get update
! sudo apt-get install tree
```

```
Hit:1 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InRelease
Hit:2 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:3 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64
InRelease
Hit:4 https://r2u.stat.illinois.edu/ubuntu jammy InRelease
Hit:5 http://archive.ubuntu.com/ubuntu jammy InRelease
Hit:6 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:7 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease
Hit:8 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:9 https://ppa.launchpadcontent.net/graphics-drivers/ppa/ubuntu jammy
InRelease
Hit:10 https://ppa.launchpadcontent.net/ubuntugis/ppa/ubuntu jammy InRelease
Reading package lists… Done
W: Skipping acquire of configured file 'main/source/Sources' as repository
'https://r2u.stat.illinois.edu/ubuntu jammy InRelease' does not seem to provide
it (sources.list entry misspelt?)
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
tree is already the newest version (2.0.2-1).
0 upgraded, 0 newly installed, 0 to remove and 31 not upgraded.
```

[ ]:
```python
#@title Imports
import nltk
from nltk.tokenize import RegexpTokenizer
import sentencepiece
import contractions
import spacy


import evaluate
from datasets import load_dataset, Dataset, DatasetDict

import torch
import torch.nn as nn
from torchinfo import summary

import transformers
from transformers import AutoTokenizer, AutoModel,␣
 ↪AutoModelForSequenceClassification, TrainingArguments, Trainer, BertConfig,␣
 ↪BertForSequenceClassification

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import sklearn
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report,␣
 ↪precision_recall_fscore_support, accuracy_score

import json
import datetime
import zoneinfo
from datetime import datetime

import math
from mpl_toolkits.mplot3d import Axes3D
from sklearn.decomposition import PCA
from scipy.stats import entropy
from sklearn.metrics import confusion_matrix
```

[ ]:
```python
# @title Mount Google Drive
```

[ ]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[ ]:
```
dir_root = '/content/drive/MyDrive/266-final/'
# dir_data = '/content/drive/MyDrive/266-final/data/'
# dir_data = '/content/drive/MyDrive/266-final/data/se21-t1-comp-lex-master/'
dir_data = '/content/drive/MyDrive/266-final/data/266-comp-lex-master'
dir_models = '/content/drive/MyDrive/266-final/models/'
dir_results = '/content/drive/MyDrive/266-final/results/'
log_filename = "experiment_runs.txt"
log_filepath = os.path.join(dir_results, log_filename)
```

[ ]:
```
wandbai_api_key = ""
```

[ ]:
```
!tree /content/drive/MyDrive/266-final/data/266-comp-lex-master/
```

```
/content/drive/MyDrive/266-final/data/266-comp-lex-master/
  fe-test-labels
      test_multi_df.csv
      test_single_df.csv
  fe-train
      train_multi_df.csv
      train_single_df.csv
  fe-trial-val
      trial_val_multi_df.csv
      trial_val_single_df.csv
  test-labels
      lcp_multi_test.tsv
      lcp_single_test.tsv
  train
      lcp_multi_train.tsv
      lcp_single_train.tsv
  trial
       lcp_multi_trial.tsv
       lcp_single_trial.tsv

6 directories, 12 files
```

[ ]:
```
!ls -R /content/drive/MyDrive/266-final/data/266-comp-lex-master/
```

```
/content/drive/MyDrive/266-final/data/266-comp-lex-master/:
fe-test-labels  fe-train  fe-trial-val  test-labels  train  trial

/content/drive/MyDrive/266-final/data/266-comp-lex-master/fe-test-labels:
test_multi_df.csv  test_single_df.csv

/content/drive/MyDrive/266-final/data/266-comp-lex-master/fe-train:
train_multi_df.csv  train_single_df.csv
```

```
/content/drive/MyDrive/266-final/data/266-comp-lex-master/fe-trial-val:
trial_val_multi_df.csv  trial_val_single_df.csv

/content/drive/MyDrive/266-final/data/266-comp-lex-master/test-labels:
lcp_multi_test.tsv  lcp_single_test.tsv

/content/drive/MyDrive/266-final/data/266-comp-lex-master/train:
lcp_multi_train.tsv  lcp_single_train.tsv

/content/drive/MyDrive/266-final/data/266-comp-lex-master/trial:
lcp_multi_trial.tsv  lcp_single_trial.tsv
```

[ ]:
```
!tree /content/drive/MyDrive/266-final/data/266-comp-lex-master/
```

```
/content/drive/MyDrive/266-final/data/266-comp-lex-master/
  fe-test-labels
      test_multi_df.csv
      test_single_df.csv
  fe-train
      train_multi_df.csv
      train_single_df.csv
  fe-trial-val
      trial_val_multi_df.csv
      trial_val_single_df.csv
  test-labels
      lcp_multi_test.tsv
      lcp_single_test.tsv
  train
      lcp_multi_train.tsv
      lcp_single_train.tsv
  trial
      lcp_multi_trial.tsv
      lcp_single_trial.tsv

6 directories, 12 files
```

[ ]:
```
#@title Import Data
```

[ ]:
```
df_names = [
    "train_single_df",
    "train_multi_df",
    "trial_val_single_df",
    "trial_val_multi_df",
    "test_single_df",
    "test_multi_df"
]

loaded_dataframes = {}
```

```python
for df_name in df_names:
    if "train" in df_name:
        subdir = "fe-train"
    elif "trial_val" in df_name:
        subdir = "fe-trial-val"
    elif "test" in df_name:
        subdir = "fe-test-labels"
    else:
        subdir = None

    if subdir:
        read_path = os.path.join(dir_data, subdir, f"{df_name}.csv")
        loaded_df = pd.read_csv(read_path)
        loaded_dataframes[df_name] = loaded_df
        print(f"Loaded {df_name} from {read_path}")

# for df_name, df in loaded_dataframes.items():
#     print(f"\n>>> {df_name} shape: {df.shape}")
#     if 'binary_complexity' in df.columns:
#         print(df['binary_complexity'].value_counts())
#         print(df.info())
#         print(df.head())

for df_name, df in loaded_dataframes.items():
    globals()[df_name] = df
    print(f"{df_name} loaded into global namespace.")
```

```
Loaded train_single_df from /content/drive/MyDrive/266-final/data/266-comp-lex-
master/fe-train/train_single_df.csv
Loaded train_multi_df from /content/drive/MyDrive/266-final/data/266-comp-lex-
master/fe-train/train_multi_df.csv
Loaded trial_val_single_df from /content/drive/MyDrive/266-final/data/266-comp-
lex-master/fe-trial-val/trial_val_single_df.csv
Loaded trial_val_multi_df from /content/drive/MyDrive/266-final/data/266-comp-
lex-master/fe-trial-val/trial_val_multi_df.csv
Loaded test_single_df from /content/drive/MyDrive/266-final/data/266-comp-lex-
master/fe-test-labels/test_single_df.csv
Loaded test_multi_df from /content/drive/MyDrive/266-final/data/266-comp-lex-
master/fe-test-labels/test_multi_df.csv
train_single_df loaded into global namespace.
train_multi_df loaded into global namespace.
trial_val_single_df loaded into global namespace.
trial_val_multi_df loaded into global namespace.
test_single_df loaded into global namespace.
test_multi_df loaded into global namespace.
```

- Functional tests pass, we can proceed with Baseline Modeling

0.2 Experiments

0.2.1 Helper Functions ** Run **

```
[ ]: MODEL_LINEAGE = {}

     def get_model_and_tokenizer(
         remote_model_name: str = None,
         local_model_path: str = None,
         config=None
     ):
         """
         Loads the model & tokenizer for classification.
         If 'local_model_path' is specified, load from that path.
         Otherwise, fall back to 'remote_model_name'.

         Optional: 'config' can be a custom BertConfig/AutoConfig object
                   to override certain configuration parameters.

         Records complete traceable lineage in the global MODEL_LINEAGE.
         """
         global MODEL_LINEAGE

         if local_model_path:
             print(f"Loading from local path: {local_model_path}")
             tokenizer = AutoTokenizer.from_pretrained(local_model_path)

             # If a config object is provided, we pass it to from_pretrained.
             # Otherwise, it just uses the config that is part of local_model_path.
             if config is not None:
                 model = AutoModelForSequenceClassification.from_pretrained(
                     local_model_path,
                     config=config
                 )
             else:
                 model = AutoModelForSequenceClassification.
       ↪from_pretrained(local_model_path)

             MODEL_LINEAGE = {
                 "type": "offline_checkpoint",
                 "path": local_model_path,
                 "timestamp": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
             }
         elif remote_model_name:
             print(f"Loading from Hugging Face model: {remote_model_name}")
             tokenizer = AutoTokenizer.from_pretrained(remote_model_name)

             if config is not None:
```

```
            model = AutoModelForSequenceClassification.from_pretrained(
                remote_model_name,
                config=config
            )
        else:
            model = AutoModelForSequenceClassification.
↪from_pretrained(remote_model_name)

        MODEL_LINEAGE = {
            "type": "huggingface_hub",
            "path": remote_model_name,
            "timestamp": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        }
    else:
        raise ValueError("You must provide either a remote_model_name or a␣
↪local_model_path!")

    return model, tokenizer
```

```
[ ]: def freeze_unfreeze_layers(model, layers_to_unfreeze=None):
         """
         Toggles requires_grad = False for all parameters
         except for those whose names contain any string in layers_to_unfreeze.
         By default, always unfreeze classifier/heads.
         """
         if layers_to_unfreeze is None:
             layers_to_unfreeze = ["classifier.", "pooler."]

         for name, param in model.named_parameters():
             if any(substring in name for substring in layers_to_unfreeze):
                 param.requires_grad = True
             else:
                 param.requires_grad = False
```

```
[ ]: def encode_examples(examples, tokenizer, text_col, max_length=256):
         """
         Tokenizes a batch of texts from 'examples[text_col]' using the given␣
↪tokenizer.
         Returns a dict with 'input_ids', 'attention_mask', etc.
         """
         texts = examples[text_col]
         encoded = tokenizer(
             texts,
             truncation=True,
             padding='max_length',
             max_length=max_length
         )
```

```
    return encoded
```

```
[ ]: def prepare_dataset(df, tokenizer, text_col, label_col, max_length=256):
         """
         Converts a Pandas DataFrame to a Hugging Face Dataset,
         then applies 'encode_examples' to tokenize.
         """
         dataset = Dataset.from_pandas(df)

         dataset = dataset.map(
             lambda batch: encode_examples(batch, tokenizer, text_col, max_length),
             batched=True
         )

         dataset = dataset.rename_column(label_col, "labels")
         dataset.set_format(type='torch',
                            columns=['input_ids', 'attention_mask', 'labels'])
         return dataset
```

```
[ ]: def compute_metrics(eval_pred):
         """
         Computes classification metrics, including accuracy, precision, recall, and␣
      ↪F1.
         """
         logits, labels = eval_pred
         preds = np.argmax(logits, axis=1)

         metric_accuracy  = evaluate.load("accuracy")
         metric_precision = evaluate.load("precision")
         metric_recall    = evaluate.load("recall")
         metric_f1        = evaluate.load("f1")

         accuracy_result  = metric_accuracy.compute(predictions=preds,␣
      ↪references=labels)
         precision_result = metric_precision.compute(predictions=preds,␣
      ↪references=labels, average="binary")
         recall_result    = metric_recall.compute(predictions=preds,␣
      ↪references=labels, average="binary")
         f1_result        = metric_f1.compute(predictions=preds, references=labels,␣
      ↪average="binary")

         return {
             "accuracy"       : accuracy_result["accuracy"],
             "precision": precision_result["precision"],
             "recall"   : recall_result["recall"],
             "f1"       : f1_result["f1"]
         }
```

```
[ ]: def gather_config_details(model):
         """
         Enumerates every attribute in model.config
         """
         config_items = {}
         for attr_name, attr_value in vars(model.config).items():
             config_items[attr_name] = attr_value
         return config_items

     def gather_model_details(model):
         """
         Extracts total layers, total params, trainable params, and activation␣
      ↪function
         from a Transformers model. Adjust logic as needed for different␣
      ↪architectures.
         """
         details = {}

         try:
             total_params = model.num_parameters()
             trainable_params = model.num_parameters(only_trainable=True)
         except AttributeError:
             all_params = list(model.parameters())
             total_params = sum(p.numel() for p in all_params)
             trainable_params = sum(p.numel() for p in all_params if p.requires_grad)

         details["model_total_params"] = total_params
         details["model_trainable_params"] = trainable_params

         if hasattr(model, "bert") and hasattr(model.bert, "pooler"):
             act_obj = getattr(model.bert.pooler, "activation", None)
             details["pooler_activation_function"] = act_obj.__class__.__name__ if␣
      ↪act_obj else "N/A"
         else:
             details["pooler_activation_function"] = "N/A"

         details["config_attributes"] = gather_config_details(model)
         return details

     def gather_all_run_metrics(trainer, train_dataset=None, val_dataset=None,␣
      ↪test_dataset=None):
         """
         Gathers final training metrics, final validation metrics, final test␣
      ↪metrics.
         Instead of only parsing the final train_loss from the log, we also do a full
         trainer.evaluate(train_dataset) to get the same set of metrics that val/
      ↪test have.
```

```
    """
    results = {}

    if train_dataset is not None:
        train_metrics = trainer.evaluate(train_dataset)
        for k, v in train_metrics.items():
            results[f"train_{k}"] = v
    else:
        results["train_metrics"] = "No train dataset provided"

    if val_dataset is not None:
        val_metrics = trainer.evaluate(val_dataset)
        for k, v in val_metrics.items():
            results[f"val_{k}"] = v
    else:
        results["val_metrics"] = "No val dataset provided"

    if test_dataset is not None:
        test_metrics = trainer.evaluate(test_dataset)
        for k, v in test_metrics.items():
            results[f"test_{k}"] = v
    else:
        results["test_metrics"] = "No test dataset provided"

    return results

# def log_experiment_results_json(experiment_meta, model_details, run_metrics,␣
 ↪log_file):
#     """
#     Logs experiment metadata, model details, and metrics to a JSON lines file.
#     Automatically concatenates the 'checkpoint_path' to the 'model_lineage'.
#     """
#     checkpoint_path = model_details.get("checkpoint_path")
#     if checkpoint_path:
#         if "model_lineage" not in model_details:
#             model_details["model_lineage"] = ""
#         if model_details["model_lineage"]:
#             model_details["model_lineage"] += " -> "
#         model_details["model_lineage"] += checkpoint_path

#     record = {
#         "timestamp": str(datetime.datetime.now()),
#         "experiment_meta": experiment_meta,
#         "model_details": model_details,
#         "run_metrics": run_metrics
#     }
#     with open(log_file, "a", encoding="utf-8") as f:
#         json.dump(record, f)
#         f.write("\n")

def log_experiment_results_json(experiment_meta, model_details, run_metrics,␣
 ↪log_file):
    """
    Logs experiment metadata, model details, and metrics to a JSON lines file.
    Automatically concatenates the 'checkpoint_path' to the 'model_lineage'
    and uses Pacific time for the timestamp.
    """
    checkpoint_path = model_details.get("checkpoint_path")
    if checkpoint_path:
        if "model_lineage" not in model_details:
            model_details["model_lineage"] = ""
        if model_details["model_lineage"]:
            model_details["model_lineage"] += " -> "
        model_details["model_lineage"] += checkpoint_path

    pacific_time = datetime.now(zoneinfo.ZoneInfo("America/Los_Angeles")) #␣
 ↪update to support pacific time
    timestamp_str = pacific_time.isoformat()

    record = {
        "timestamp": timestamp_str,
        "experiment_meta": experiment_meta,
        "model_details": model_details,
        "run_metrics": run_metrics
    }

    with open(log_file, "a", encoding="utf-8") as f:
        json.dump(record, f)
        f.write("\n")
```

0.2.2 Experiment Cohort Design

[ ]:
```
# Define Experiment Parameters

named_model = "bert-base-cased"
# named_model = "roberta-base"
# named_model = "bert-large"
# named_model = "roberta-large"
# named_model = "" # modern bert

# learning_rate = 1e-3
# learning_rate = 1e-4
learning_rate = 1e-5
```

```
# learning_rate = 5e-6
# learning_rate = 5e-7
# learning_rate = 5e-8

# num_epochs = 1
# num_epochs = 3
# num_epochs = 5
num_epochs = 25
# num_epochs = 15
# num_epochs = 20

# length_max = 128
length_max = 256
# length_max = 348
# length_max = 512

# size_batch = 1
# size_batch = 4
# size_batch = 8
size_batch = 16
# size_batch = 24
# size_batch = 32
# size_batch = 64
# size_batch = 128

# regularization_weight_decay = 0
regularization_weight_decay = 0.1
# regularization_weight_decay = 0.5

y_col = "binary_complexity"
# y_col = "complexity"

x_task = "single"
# x_task = "multi"

# x_col = "sentence"
x_col = "sentence_no_contractions"
# x_col = "pos_sequence"
# x_col = "dep_sequence"
# x_col = "morph_sequence"

if x_task == "single":
    df_train = train_single_df
    df_val   = trial_val_single_df
    df_test  = test_single_df
else:
    df_train = train_multi_df
    df_val   = trial_val_multi_df
    df_test  = test_multi_df

custom_config = BertConfig.from_pretrained("bert-base-cased")

custom_config.hidden_dropout_prob = 0.1
# custom_config.intermediate_size = 3072
# custom_config.intermediate_size = 6144
# custom_config.num_attention_heads = 12
# custom_config.num_hidden_layers = 12
custom_config.gradient_checkpointing = False
custom_config.attention_probs_dropout_prob = 0.1
# custom_config.max_position_embeddings = 512
# custom_config.type_vocab_size = 2
custom_config.hidden_act = "gelu"  # alts: "relu" "silu"
# custom_config.vocab_size = 28996  # must match

# model.bert.pooler.activation = nn.ReLU() # Tanh() replaced as the pooler␣
 ↪layer activation function in side-by-side with 1.1
```

```
[ ]: def train_transformer_model(
         model,
         tokenizer,
         train_dataset,
         val_dataset,
         output_dir=dir_results,
         num_epochs=num_epochs,
         batch_size=size_batch,
         lr=learning_rate,
         weight_decay=regularization_weight_decay
     ):
         """
         Sets up a Trainer and trains the model for 'num_epochs' using the given␣
      ↪dataset.
         Returns the trained model and the Trainer object for possible re-use or␣
      ↪analysis.
         """

         training_args = TrainingArguments(
             output_dir=output_dir,
             num_train_epochs=num_epochs,
             per_device_train_batch_size=batch_size,
             per_device_eval_batch_size=batch_size,
             evaluation_strategy="epoch",
             save_strategy="no",
             logging_strategy="epoch",
             learning_rate=lr,
             weight_decay=weight_decay,
             report_to=["none"],  # or "wandb"
             warmup_steps=1
         )

         trainer = Trainer(
             model=model,
             args=training_args,
             train_dataset=train_dataset,
             eval_dataset=val_dataset,
             tokenizer=tokenizer,  # optional
             compute_metrics=compute_metrics
         )

         trainer.train()
         return model, trainer
```

Model Inspection ** **Run** **

```
[ ]: print("model checkpoints:", dir_models)
     # !ls /content/drive/MyDrive/266-final/models/
```

```
model checkpoints: /content/drive/MyDrive/266-final/models/
```

```
[ ]: # Load Model & Tokenizer
     # model, tokenizer = get_model_and_tokenizer(named_model) # deprecated argument␣
      ↪structure
     # model, tokenizer = get_model_and_tokenizer("/content/drive/MyDrive/266-final/
      ↪models/....") # proposed argument usage for checkpointed models

     # for name, param in model.named_parameters():
     #     print(name)

     model, tokenizer = get_model_and_tokenizer(
         remote_model_name="bert-base-cased",
         local_model_path=None,
         config=custom_config
     )

     # model, tokenizer = get_model_and_tokenizer(
     #     local_model_path="my_local_bert_path",
     #     config=custom_config
     # )

     print("=============")
```

```python
print(named_model, ":")
print("=============")
# print(model)
print("=============")
print(model.config)
print("=============")
print("num_parameters:", model.num_parameters())
print("=============")
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
```

```
Loading from Hugging Face model: bert-base-cased

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

=============
bert-base-cased :
=============
=============
BertConfig {
  "_attn_implementation_autoset": true,
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "torch_dtype": "float32",
  "transformers_version": "4.50.3",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 28996
}
```

```
=============
num_parameters: 108311810
=============
num_trainable_parameters: 108311810
```

Layer Configuration ** Run **

```
[ ]: # Freeze/Unfreeze Layers & Additional Activation Function Configuration

layers_to_unfreeze = [
    # "bert.embeddings.",
    # "bert.encoder.layer.0.",
    # "bert.encoder.layer.1.",
    "bert.encoder.layer.8.",
    "bert.encoder.layer.9.",
    "bert.encoder.layer.10.",
    "bert.encoder.layer.11.",
    "bert.pooler.",
    "classifier.",
]

freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)

for name, param in model.named_parameters():
    print(name, "requires_grad=", param.requires_grad)

print("\nLayers that are 'True' are trainable. 'False' are frozen.")

print("=============")
print(named_model, ":")
print("=============")
# print(model)
print("=============")
print(model.config)
print("=============")
print("num_parameters:", model.num_parameters())
print("=============")
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
```

```
bert.embeddings.word_embeddings.weight requires_grad= False
bert.embeddings.position_embeddings.weight requires_grad= False
bert.embeddings.token_type_embeddings.weight requires_grad= False
bert.embeddings.LayerNorm.weight requires_grad= False
bert.embeddings.LayerNorm.bias requires_grad= False
bert.encoder.layer.0.attention.self.query.weight requires_grad= False
bert.encoder.layer.0.attention.self.query.bias requires_grad= False
bert.encoder.layer.0.attention.self.key.weight requires_grad= False
bert.encoder.layer.0.attention.self.key.bias requires_grad= False
bert.encoder.layer.0.attention.self.value.weight requires_grad= False
bert.encoder.layer.0.attention.self.value.bias requires_grad= False
bert.encoder.layer.0.attention.output.dense.weight requires_grad= False
bert.encoder.layer.0.attention.output.dense.bias requires_grad= False
bert.encoder.layer.0.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.0.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.0.intermediate.dense.weight requires_grad= False
bert.encoder.layer.0.intermediate.dense.bias requires_grad= False
bert.encoder.layer.0.output.dense.weight requires_grad= False
bert.encoder.layer.0.output.dense.bias requires_grad= False
bert.encoder.layer.0.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.0.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.1.attention.self.query.weight requires_grad= False
bert.encoder.layer.1.attention.self.query.bias requires_grad= False
bert.encoder.layer.1.attention.self.key.weight requires_grad= False
bert.encoder.layer.1.attention.self.key.bias requires_grad= False
bert.encoder.layer.1.attention.self.value.weight requires_grad= False
bert.encoder.layer.1.attention.self.value.bias requires_grad= False
bert.encoder.layer.1.attention.output.dense.weight requires_grad= False
bert.encoder.layer.1.attention.output.dense.bias requires_grad= False
bert.encoder.layer.1.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.1.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.1.intermediate.dense.weight requires_grad= False
bert.encoder.layer.1.intermediate.dense.bias requires_grad= False
bert.encoder.layer.1.output.dense.weight requires_grad= False
bert.encoder.layer.1.output.dense.bias requires_grad= False
bert.encoder.layer.1.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.1.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.2.attention.self.query.weight requires_grad= False
bert.encoder.layer.2.attention.self.query.bias requires_grad= False
bert.encoder.layer.2.attention.self.key.weight requires_grad= False
bert.encoder.layer.2.attention.self.key.bias requires_grad= False
bert.encoder.layer.2.attention.self.value.weight requires_grad= False
bert.encoder.layer.2.attention.self.value.bias requires_grad= False
bert.encoder.layer.2.attention.output.dense.weight requires_grad= False
bert.encoder.layer.2.attention.output.dense.bias requires_grad= False
bert.encoder.layer.2.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.2.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.2.intermediate.dense.weight requires_grad= False
bert.encoder.layer.2.intermediate.dense.bias requires_grad= False
bert.encoder.layer.2.output.dense.weight requires_grad= False
bert.encoder.layer.2.output.dense.bias requires_grad= False
bert.encoder.layer.2.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.2.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.3.attention.self.query.weight requires_grad= False
bert.encoder.layer.3.attention.self.query.bias requires_grad= False
bert.encoder.layer.3.attention.self.key.weight requires_grad= False
bert.encoder.layer.3.attention.self.key.bias requires_grad= False
bert.encoder.layer.3.attention.self.value.weight requires_grad= False
bert.encoder.layer.3.attention.self.value.bias requires_grad= False
bert.encoder.layer.3.attention.output.dense.weight requires_grad= False
bert.encoder.layer.3.attention.output.dense.bias requires_grad= False
bert.encoder.layer.3.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.3.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.3.intermediate.dense.weight requires_grad= False
bert.encoder.layer.3.intermediate.dense.bias requires_grad= False
bert.encoder.layer.3.output.dense.weight requires_grad= False
bert.encoder.layer.3.output.dense.bias requires_grad= False
bert.encoder.layer.3.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.3.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.4.attention.self.query.weight requires_grad= False
bert.encoder.layer.4.attention.self.query.bias requires_grad= False
bert.encoder.layer.4.attention.self.key.weight requires_grad= False
bert.encoder.layer.4.attention.self.key.bias requires_grad= False
bert.encoder.layer.4.attention.self.value.weight requires_grad= False
bert.encoder.layer.4.attention.self.value.bias requires_grad= False
bert.encoder.layer.4.attention.output.dense.weight requires_grad= False
bert.encoder.layer.4.attention.output.dense.bias requires_grad= False
bert.encoder.layer.4.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.4.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.4.intermediate.dense.weight requires_grad= False
bert.encoder.layer.4.intermediate.dense.bias requires_grad= False
bert.encoder.layer.4.output.dense.weight requires_grad= False
bert.encoder.layer.4.output.dense.bias requires_grad= False
bert.encoder.layer.4.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.4.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.5.attention.self.query.weight requires_grad= False
bert.encoder.layer.5.attention.self.query.bias requires_grad= False
bert.encoder.layer.5.attention.self.key.weight requires_grad= False
bert.encoder.layer.5.attention.self.key.bias requires_grad= False
bert.encoder.layer.5.attention.self.value.weight requires_grad= False
bert.encoder.layer.5.attention.self.value.bias requires_grad= False
bert.encoder.layer.5.attention.output.dense.weight requires_grad= False
bert.encoder.layer.5.attention.output.dense.bias requires_grad= False
bert.encoder.layer.5.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.5.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.5.intermediate.dense.weight requires_grad= False
bert.encoder.layer.5.intermediate.dense.bias requires_grad= False
bert.encoder.layer.5.output.dense.weight requires_grad= False
bert.encoder.layer.5.output.dense.bias requires_grad= False
bert.encoder.layer.5.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.5.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.6.attention.self.query.weight requires_grad= False
bert.encoder.layer.6.attention.self.query.bias requires_grad= False
bert.encoder.layer.6.attention.self.key.weight requires_grad= False
bert.encoder.layer.6.attention.self.key.bias requires_grad= False
bert.encoder.layer.6.attention.self.value.weight requires_grad= False
bert.encoder.layer.6.attention.self.value.bias requires_grad= False
bert.encoder.layer.6.attention.output.dense.weight requires_grad= False
bert.encoder.layer.6.attention.output.dense.bias requires_grad= False
bert.encoder.layer.6.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.6.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.6.intermediate.dense.weight requires_grad= False
bert.encoder.layer.6.intermediate.dense.bias requires_grad= False
bert.encoder.layer.6.output.dense.weight requires_grad= False
bert.encoder.layer.6.output.dense.bias requires_grad= False
bert.encoder.layer.6.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.6.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.7.attention.self.query.weight requires_grad= False
bert.encoder.layer.7.attention.self.query.bias requires_grad= False
bert.encoder.layer.7.attention.self.key.weight requires_grad= False
bert.encoder.layer.7.attention.self.key.bias requires_grad= False
bert.encoder.layer.7.attention.self.value.weight requires_grad= False
bert.encoder.layer.7.attention.self.value.bias requires_grad= False
bert.encoder.layer.7.attention.output.dense.weight requires_grad= False
bert.encoder.layer.7.attention.output.dense.bias requires_grad= False
bert.encoder.layer.7.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.7.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.7.intermediate.dense.weight requires_grad= False
bert.encoder.layer.7.intermediate.dense.bias requires_grad= False
bert.encoder.layer.7.output.dense.weight requires_grad= False
bert.encoder.layer.7.output.dense.bias requires_grad= False
bert.encoder.layer.7.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.7.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.8.attention.self.query.weight requires_grad= True
bert.encoder.layer.8.attention.self.query.bias requires_grad= True
bert.encoder.layer.8.attention.self.key.weight requires_grad= True
bert.encoder.layer.8.attention.self.key.bias requires_grad= True
bert.encoder.layer.8.attention.self.value.weight requires_grad= True
bert.encoder.layer.8.attention.self.value.bias requires_grad= True
bert.encoder.layer.8.attention.output.dense.weight requires_grad= True
bert.encoder.layer.8.attention.output.dense.bias requires_grad= True
bert.encoder.layer.8.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.8.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.8.intermediate.dense.weight requires_grad= True
bert.encoder.layer.8.intermediate.dense.bias requires_grad= True
bert.encoder.layer.8.output.dense.weight requires_grad= True
bert.encoder.layer.8.output.dense.bias requires_grad= True
bert.encoder.layer.8.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.8.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.9.attention.self.query.weight requires_grad= True
bert.encoder.layer.9.attention.self.query.bias requires_grad= True
bert.encoder.layer.9.attention.self.key.weight requires_grad= True
bert.encoder.layer.9.attention.self.key.bias requires_grad= True
bert.encoder.layer.9.attention.self.value.weight requires_grad= True
bert.encoder.layer.9.attention.self.value.bias requires_grad= True
bert.encoder.layer.9.attention.output.dense.weight requires_grad= True
bert.encoder.layer.9.attention.output.dense.bias requires_grad= True
bert.encoder.layer.9.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.9.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.9.intermediate.dense.weight requires_grad= True
bert.encoder.layer.9.intermediate.dense.bias requires_grad= True
bert.encoder.layer.9.output.dense.weight requires_grad= True
bert.encoder.layer.9.output.dense.bias requires_grad= True
bert.encoder.layer.9.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.9.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.10.attention.self.query.weight requires_grad= True
bert.encoder.layer.10.attention.self.query.bias requires_grad= True
bert.encoder.layer.10.attention.self.key.weight requires_grad= True
bert.encoder.layer.10.attention.self.key.bias requires_grad= True
bert.encoder.layer.10.attention.self.value.weight requires_grad= True
bert.encoder.layer.10.attention.self.value.bias requires_grad= True
bert.encoder.layer.10.attention.output.dense.weight requires_grad= True
bert.encoder.layer.10.attention.output.dense.bias requires_grad= True
bert.encoder.layer.10.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.10.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.10.intermediate.dense.weight requires_grad= True
bert.encoder.layer.10.intermediate.dense.bias requires_grad= True
bert.encoder.layer.10.output.dense.weight requires_grad= True
bert.encoder.layer.10.output.dense.bias requires_grad= True
bert.encoder.layer.10.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.10.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.11.attention.self.query.weight requires_grad= True
bert.encoder.layer.11.attention.self.query.bias requires_grad= True
bert.encoder.layer.11.attention.self.key.weight requires_grad= True
bert.encoder.layer.11.attention.self.key.bias requires_grad= True
bert.encoder.layer.11.attention.self.value.weight requires_grad= True
bert.encoder.layer.11.attention.self.value.bias requires_grad= True
bert.encoder.layer.11.attention.output.dense.weight requires_grad= True
bert.encoder.layer.11.attention.output.dense.bias requires_grad= True
bert.encoder.layer.11.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.11.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.11.intermediate.dense.weight requires_grad= True
bert.encoder.layer.11.intermediate.dense.bias requires_grad= True
bert.encoder.layer.11.output.dense.weight requires_grad= True
bert.encoder.layer.11.output.dense.bias requires_grad= True
bert.encoder.layer.11.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.11.output.LayerNorm.bias requires_grad= True
bert.pooler.dense.weight requires_grad= True
bert.pooler.dense.bias requires_grad= True
classifier.weight requires_grad= True
classifier.bias requires_grad= True
```

```
Layers that are 'True' are trainable. 'False' are frozen.
=============
bert-base-cased :
=============
=============
BertConfig {
  "_attn_implementation_autoset": true,
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "torch_dtype": "float32",
  "transformers_version": "4.50.3",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 28996
}

=============
num_parameters: 108311810
=============
num_trainable_parameters: 28943618
```

Dataset Preparation ** Run **

[ ]:
```
# Tokenize & Prepare Datasets

train_data_hf = prepare_dataset(
    df_train,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max
)
```

```
val_data_hf = prepare_dataset(
    df_val,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max
)

test_data_hf = prepare_dataset(
    df_test,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max

)

print("Datasets prepared. Sample from train_data_hf:\n", train_data_hf[10])
# print("Datasets prepared. Sample from train_data_hf:\n", val_data_hf[10])
# print("Datasets prepared. Sample from train_data_hf:\n", test_data_hf[10])
```

```
Map:   0%|          | 0/7000 [00:00<?, ? examples/s]

Map:   0%|          | 0/1000 [00:00<?, ? examples/s]

Map:   0%|          | 0/1000 [00:00<?, ? examples/s]

Datasets prepared. Sample from train_data_hf:
 {'labels': tensor(0), 'input_ids': tensor([  101,  1130,  1864,   117,  1175,
1110,   170,  6145,  4423,  1103,
        10838,  1104,  1103,  1177,   118,  1270,  6298,  4692,   117,  1216,
         1112,  1343,  2272,  1106,  3750,   117,  1154,  1103,  1812,  7216,
          119,   102,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
               0,      0,      0,      0,      0,      0,      0,      0,      0,      0,
               0,      0,      0,      0,      0,      0,      0,      0,      0,      0,
               0,      0,      0,      0,      0,      0,      0,      0,      0,      0,
               0,      0,      0,      0,      0,      0,      0,      0,      0,      0,
               0,      0,      0,      0,      0,      0,      0,      0,      0,      0,
               0,      0,      0,      0,      0,      0,      0,      0,      0,      0,
               0,      0,      0,      0,      0,      0]), 'attention_mask': tensor([1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])}
```

0.2.3 bert-base-cased

[ ]:
```
# Define Experiment Parameters
named_model = "bert-base-cased"
# named_model = "roberta-base"
# named_model = "bert-large"
# named_model = "roberta-large"
# named_model = "" # modern bert
############
regularization_weight_decay = 0.5
learning_rate = 5e-6
size_batch = 128
length_max = 128
num_epochs = 1
###########################################
# x_col = "sentence"
x_col = "sentence_no_contractions"
# x_col = "pos_sequence"
# x_col = "dep_sequence"
# x_col = "morph_sequence"
# x_col = "snc_pos_seq"
# x_col = "snc_pos_alt"
# x_col = "snc_morph_seq"
# x_col = "snc_morph_alt"
# x_col = "snc_dep_seq"
# x_col = "snc_dep_alt"
# x_col = "snc_morph_complexity_value"
############
```

```
y_col = "binary_complexity"
# y_col = "complexity"
############
# x_task = "single"
x_task = "multi"
if x_task == "single":
    df_train = train_single_df
    df_val   = trial_val_single_df
    df_test  = test_single_df
else:
    df_train = train_multi_df
    df_val   = trial_val_multi_df
    df_test  = test_multi_df
##########################################
# Tokenize & Prepare Datasets
train_data_hf = prepare_dataset(
    df_train,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
val_data_hf = prepare_dataset(
    df_val,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
test_data_hf = prepare_dataset(
    df_test,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
print("Datasets prepared. Sample from train_data_hf:\n", train_data_hf[10])
# print("Datasets prepared. Sample from train_data_hf:\n", val_data_hf[10])
# print("Datasets prepared. Sample from train_data_hf:\n", test_data_hf[10])
##########################################
# custom_config = BertConfig.from_pretrained("bert-base-cased")
# custom_config.hidden_act = "gelu"  # alts: "relu" "silu"
# custom_config.attention_probs_dropout_prob = 0.1
# custom_config.hidden_dropout_prob = 0.1
# custom_config.gradient_checkpointing = False
##########################################
model, tokenizer = get_model_and_tokenizer(
    remote_model_name="bert-base-cased",
    local_model_path=None,
    config=None)
```

```python
############
# model, tokenizer = get_model_and_tokenizer(
#     remote_model_name=None
#     local_model_path="...CONFIGURE_PATH...",
#     config=custom_config)
print("=============")
print(named_model, ":")
print("=============")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters at load:", model.
 ↪num_parameters(only_trainable=True))
print("=============")
print("model lineage:", MODEL_LINEAGE)
print("=============")
##########################################
layers_to_unfreeze = [
    # "bert.embeddings.",
    # "bert.encoder.layer.0.",
    # "bert.encoder.layer.1.",
    # "bert.encoder.layer.8.",
    # "bert.encoder.layer.9.",
    # "bert.encoder.layer.10.",
    "bert.encoder.layer.11.",
    "bert.pooler.",
    "classifier.",
]
freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)
print(model.config)
print("=============")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
print("=============")
##########################################
print("Experiment configuration used with this experiment:")
print("model used:", named_model)
print("learning rate used:", learning_rate)
print("number of epochs:", num_epochs)
print("maximum sequence length:", length_max)
print("batch size used:", size_batch)
print("regularization value:", regularization_weight_decay)
print("outcome variable:", y_col)
print("task:", x_task)
print("input column:", x_col)
```

```
Map:   0%|          | 0/1300 [00:00<?, ? examples/s]

Map:   0%|          | 0/250 [00:00<?, ? examples/s]

Map:   0%|          | 0/250 [00:00<?, ? examples/s]
```

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

Datasets prepared. Sample from train_data_hf:
 {'labels': tensor(0), 'input_ids': tensor([  101,  1109,  1210,  1808,  1104,
4836, 15949,   117,  1114,  1167,
         1190,   123,  1288,  8724,  1105, 24649,   117,  3657,  1107,  1103,
          163,  4626,  2758,  3293, 11171,   117,  1134, 11228, 11363,  1103,
         1352,  1104,  1103,  5655,  1206,  2470,  1386,  1105, 10585,   119,
          102,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0]),
'attention_mask': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0])}
Loading from Hugging Face model: bert-base-cased
=============
bert-base-cased :
=============
num_parameters: 108311810
num_trainable_parameters at load: 108311810
=============
model lineage: {'type': 'huggingface_hub', 'path': 'bert-base-cased',
'timestamp': '2025-04-14 01:53:36'}
=============
BertConfig {
  "_attn_implementation_autoset": true,
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "torch_dtype": "float32",
  "transformers_version": "4.50.3",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 28996
}

=============
num_parameters: 108311810
num_trainable_parameters: 7680002
=============
Experiment configuration used with this experiment:
model used: bert-base-cased
learning rate used: 5e-06
number of epochs: 1
maximum sequence length: 128
batch size used: 128
regularization value: 0.5
outcome variable: binary_complexity
task: multi
input column: sentence_no_contractions
```

[ ]:
```
# Train & Evaluate
trained_model, trainer_obj = train_transformer_model(
    model = model,
    tokenizer = tokenizer,
    train_dataset = train_data_hf,
    val_dataset = val_data_hf,
    output_dir = dir_results,
    num_epochs = num_epochs,
    batch_size = size_batch,
    lr = learning_rate,
    weight_decay = regularization_weight_decay)
metrics = trainer_obj.evaluate()
print("Validation metrics:", metrics)
test_metrics = trainer_obj.evaluate(test_data_hf) if test_data_hf else None
print("Test metrics:", test_metrics)
```

```
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1611:
FutureWarning: `evaluation_strategy` is deprecated and will be removed in
version 4.46 of  Transformers. Use `eval_strategy` instead
  warnings.warn(
<ipython-input-20-81222a87c90b>:31: FutureWarning: `tokenizer` is deprecated and
will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class`
instead.
  trainer = Trainer(

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Validation metrics: {'eval_loss': 0.71732497215271, 'eval_accuracy': 0.436,
'eval_precision': 0.43373493975903615, 'eval_recall': 1.0, 'eval_f1':
0.6050420168067226, 'eval_runtime': 1.4688, 'eval_samples_per_second': 170.205,
'eval_steps_per_second': 1.362, 'epoch': 1.0}
Test metrics: {'eval_loss': 0.6920300722122192, 'eval_accuracy': 0.508,
'eval_precision': 0.508, 'eval_recall': 1.0, 'eval_f1': 0.6737400530503979,
'eval_runtime': 1.8379, 'eval_samples_per_second': 136.027,
'eval_steps_per_second': 1.088, 'epoch': 1.0}
```

[ ]:
```
# save model checkpoint
# timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
pacific_time = datetime.now(zoneinfo.ZoneInfo("America/Los_Angeles"))
timestamp = pacific_time.isoformat()
model_save_path = os.path.join(dir_models,␣
 ↪f"{x_task}_{named_model}_{y_col}_{timestamp}")
trainer_obj.save_model(model_save_path)
print(f"Model checkpoint saved to: {model_save_path}")
# log experiment results
experiment_info = {
    "model_name": named_model,
    "learning_rate": learning_rate,
    "epochs": num_epochs,
    "batch_size": size_batch,
    "weight_decay": regularization_weight_decay,
    "x_task": x_task,
    "x_col": x_col,
    "y_col": y_col,
    "layers_to_unfreeze": layers_to_unfreeze}
model_info = gather_model_details(trained_model)
all_run_metrics = gather_all_run_metrics(
    trainer=trainer_obj,
    train_dataset=train_data_hf,
    val_dataset=val_data_hf,
    test_dataset=test_data_hf)
log_experiment_results_json(
    experiment_meta=experiment_info,
    model_details=model_info,
    run_metrics=all_run_metrics,
    log_file=log_filepath)
print(f"EXPERIMENT LOGGED TO: {log_filepath}")
```

```
Model checkpoint saved to: /content/drive/MyDrive/266-final/models/multi_bert-
base-cased_binary_complexity_2025-04-13T18:53:44.827658-07:00

<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:
/content/drive/MyDrive/266-final/results/experiment_runs.txt
```

[ ]:
```
prediction_output = trainer_obj.predict(test_data_hf)
raw_predictions = prediction_output.predictions
true_labels = prediction_output.label_ids
preds = np.argmax(raw_predictions, axis=1)
mismatch_indices = np.where(preds != true_labels)[0]

error_rows = []
for idx in mismatch_indices:
    text_value = df_test.iloc[idx][x_col]
    true_label_val = true_labels[idx]
    pred_label_val = preds[idx]

    error_rows.append({
        "hf_index": idx,
        "text": text_value,
        "true_label": true_label_val,
        "predicted_label": pred_label_val
    })

error_df = pd.DataFrame(error_rows)
df_test_for_merge = df_test.copy()
df_test_for_merge["error_matching_prefix"] = df_test_for_merge[x_col].str[:50]
# df_test_for_merge.drop(columns=[x_col], inplace=True)

error_df["error_matching_prefix"] = error_df["text"].str[:50]
error_df = error_df.merge(
    df_test_for_merge,
    on="error_matching_prefix",
    how="left",
    suffixes=("", "_source"))

error_df.to_csv("bert-base-cased_mismatches.csv", index=False)

# print("Number of misclassified samples:", len(error_df))
print("\nMerged error_df with extra columns:")
```

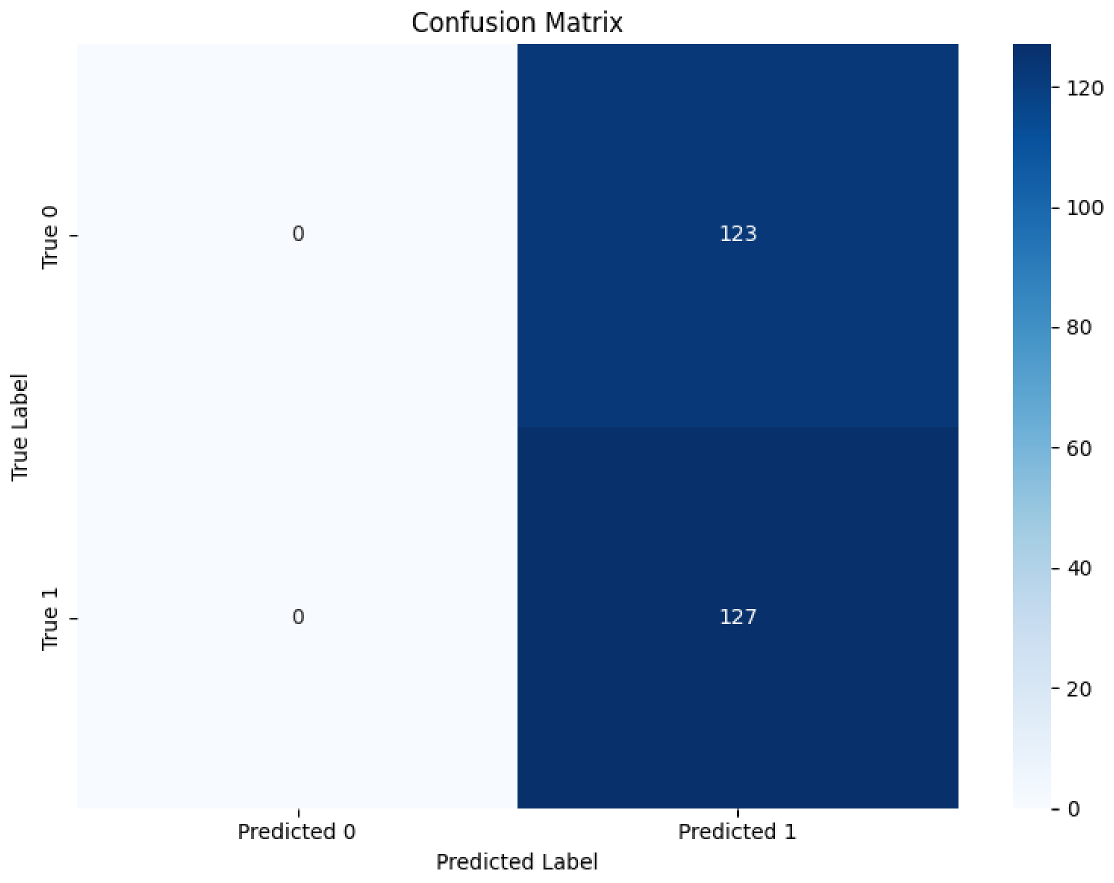
```
# display(error_df.head(15))

# print("\nConfusion Matrix:")
# cm = confusion_matrix(true_labels, preds)
# plt.figure(figsize=(8, 6))
# sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True,
#             xticklabels=["Predicted 0", "Predicted 1"],
#             yticklabels=["True 0", "True 1"])
# plt.xlabel('Predicted Label')
# plt.ylabel('True Label')
# plt.title('Confusion Matrix')
# plt.tight_layout()
# plt.show()
# print("confusion matrix metrics: \n", cm)
# error_save_path = os.path.join(dir_results,␣
 ↪f"{x_task}_{named_model}_{y_col}_{x_col}_errors.csv")
# error_df.to_csv(error_save_path, index=False)
# print("Result saved to results directory.")
prediction_output = trainer_obj.predict(test_data_hf)
raw_predictions = prediction_output.predictions
true_labels = prediction_output.label_ids
preds = np.argmax(raw_predictions, axis=1)
df_test["true_label"] = true_labels
df_test["predicted_label"] = preds
df_test["is_incorrect"] = (df_test["predicted_label"] != df_test["true_label"])
df_test["avg_embedding"] = None
device = next(model.parameters()).device
for i in range(len(df_test)):
    text_value = df_test.iloc[i][x_col]
    e = tokenizer(text_value, return_tensors="pt", truncation=True,␣
 ↪max_length=512).to(device)
    with torch.no_grad():
        emb = model.bert.embeddings.word_embeddings(e["input_ids"]).mean(dim=1).
 ↪squeeze().cpu().numpy()
    df_test.at[i,"avg_embedding"] = emb
cm = confusion_matrix(df_test["true_label"], df_test["predicted_label"])
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True,␣
 ↪xticklabels=["Predicted 0","Predicted 1"], yticklabels=["True 0","True 1"])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.tight_layout()
plt.show()
# print("confusion matrix metrics:\n", cm)

df_plot = df_test.dropna(subset=["avg_embedding"]).copy()
```

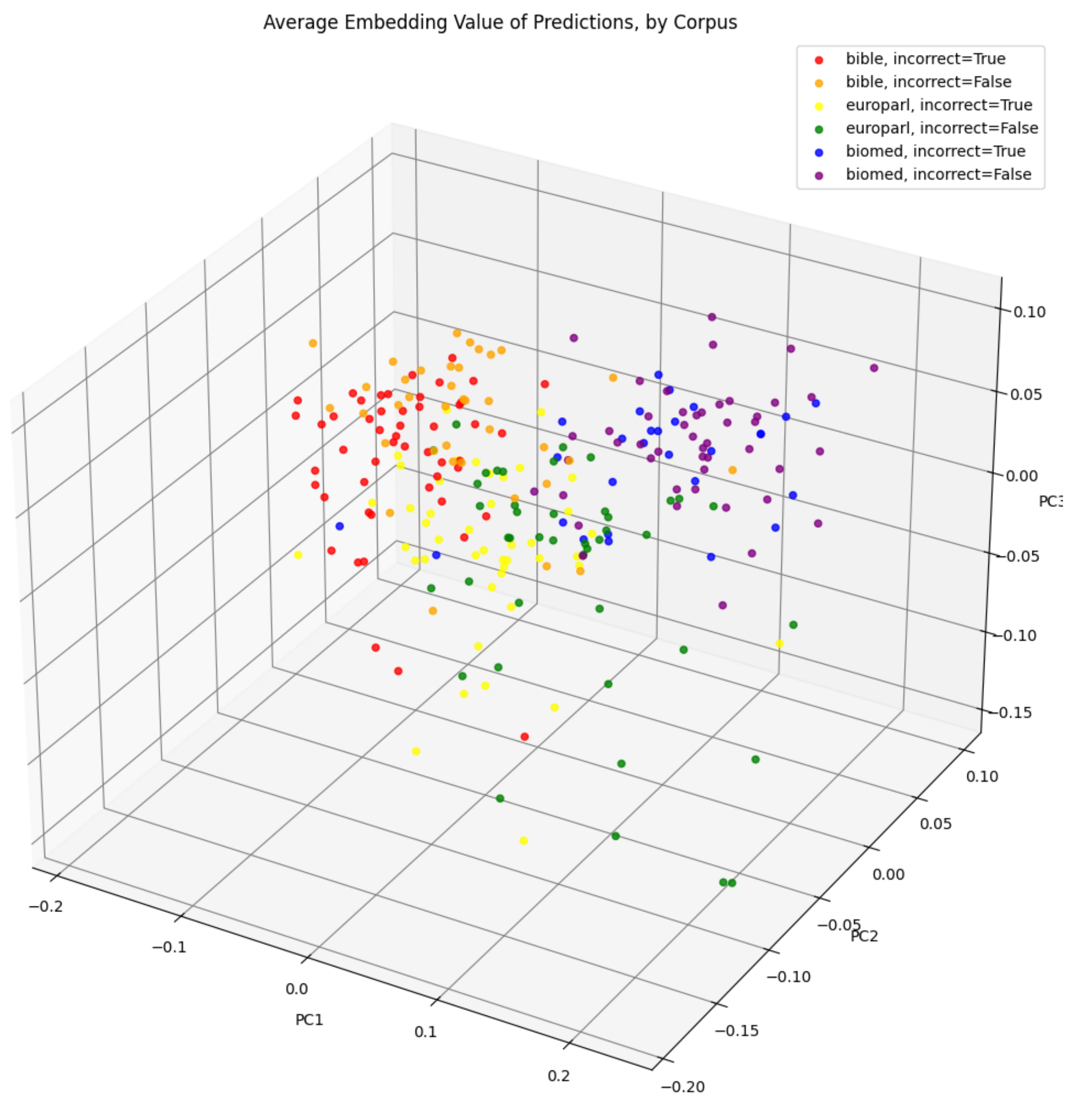
```
embeddings = np.stack(df_plot["avg_embedding"].values)
pca = PCA(n_components=3)
reduced = pca.fit_transform(embeddings)
df_plot["pca_x"] = reduced[:,0]
df_plot["pca_y"] = reduced[:,1]
df_plot["pca_z"] = reduced[:,2]
colors = {
    ("bible", True): "red",
    ("bible", False): "orange",
    ("europarl", True): "yellow",
    ("europarl", False): "green",
    ("biomed", True): "blue",
    ("biomed", False): "purple"}
fig = plt.figure(figsize=(13,13))
ax = fig.add_subplot(111, projection='3d')
fig.set_facecolor("white")
ax.set_facecolor("white")
ax.xaxis._axinfo["grid"]["color"] = "gray"
ax.yaxis._axinfo["grid"]["color"] = "gray"
ax.zaxis._axinfo["grid"]["color"] = "gray"
for (corp, incorr), color in colors.items():
    subset = df_plot[(df_plot["corpus"]==corp) &␣
  ↪(df_plot["is_incorrect"]==incorr)]
    ax.scatter(subset["pca_x"], subset["pca_y"], subset["pca_z"], c=color,␣
  ↪s=20, alpha=0.8, label=f"{corp}, incorrect={incorr}")
ax.set_title("Average Embedding Value of Predictions, by Corpus", color="black")
ax.set_xlabel("PC1", color="black")
ax.set_ylabel("PC2", color="black")
ax.set_zlabel("PC3", color="black")
ax.legend(loc="best")
plt.show()
if "corpus" in df_test.columns:
    freqs = df_test["corpus"].value_counts()
    print("\nFrequency counts of corpus:", freqs)
    err_df = df_test[df_test["is_incorrect"]==True]
    corr_df = df_test[df_test["is_incorrect"]==False]
    err_counts = err_df["corpus"].value_counts()
    corr_counts = corr_df["corpus"].value_counts()
    print("\nCounts of corpus in misclassified:", err_counts)
    print("\nCounts of corpus in correctly classified:", corr_counts)
    print("\nProportions of corpus in misclassified:", err_counts/err_counts.
  ↪sum())
    print("\nProportions of corpus in correctly classified:", corr_counts/
  ↪corr_counts.sum())
    grouped_all = df_test.groupby("corpus")["avg_embedding"].apply(lambda x: np.
  ↪mean(np.stack(x.values), axis=0))
```

```
    grouped_err = err_df.groupby("corpus")["avg_embedding"].apply(lambda x: np.
 ↪mean(np.stack(x.values), axis=0))
    grouped_corr = corr_df.groupby("corpus")["avg_embedding"].apply(lambda x:␣
 ↪np.mean(np.stack(x.values), axis=0))
    # print("\nAvg embedding of each subcorpus overall:", grouped_all)
    # print("\nAvg embedding of each subcorpus misclassified:", grouped_err)
    # print("\nAvg embedding of each subcorpus correctly classified:",␣
 ↪grouped_corr)
err_stack = np.stack(df_test[df_test["is_incorrect"]==True]["avg_embedding"].
 ↪values)
corr_stack = np.stack(df_test[df_test["is_incorrect"]==False]["avg_embedding"].
 ↪values)
# print("\nAvg embedding of records predicted incorrectly:", err_stack.
 ↪mean(axis=0))
# print("Avg embedding of records predicted correctly:", corr_stack.
 ↪mean(axis=0))
error_df = df_test[df_test["is_incorrect"]==True].copy()
error_df.to_csv("misclassified_with_all_columns.csv", index=False)
error_save_path = os.path.join(dir_results,␣
 ↪f"{x_task}_{named_model}_{y_col}_{x_col}_errors.csv")
df_test.to_csv(error_save_path, index=False)
for corp in df_test["corpus"].unique():
    subset = df_test[df_test["corpus"]==corp]
    emb_true = subset[subset["is_incorrect"]==False]["avg_embedding"]
    emb_false = subset[subset["is_incorrect"]==True]["avg_embedding"]
    if len(emb_true)==0 or len(emb_false)==0:
        print(f"No valid data for subcorpus '{corp}'")
        continue
    p = np.mean(np.stack(emb_true.values), axis=0)
    q = np.mean(np.stack(emb_false.values), axis=0)
    p_exp = np.exp(p - np.max(p))
    q_exp = np.exp(q - np.max(q))
    p_sum = p_exp.sum()
    q_sum = q_exp.sum()
    if p_sum<=0 or q_sum<=0:
        print(f"Cannot form valid distributions for subcorpus '{corp}'")
        continue
    p_dist = p_exp / p_sum
    q_dist = q_exp / q_sum
    kl_pq = entropy(p_dist, q_dist)
    kl_qp = entropy(q_dist, p_dist)
    kl_sym = 0.5*(kl_pq + kl_qp)
    print(f"Subcorpus '{corp}' symmetric KL divergence: {kl_sym}")
error_save_path = os.path.join(dir_results,␣
 ↪f"{x_task}_{named_model}_{y_col}_{x_col}_errors.csv")
df_test.to_csv(error_save_path, index=False)
```

```
print("model results saved")
```

```
<IPython.core.display.HTML object>


Merged error_df with extra columns:

<IPython.core.display.HTML object>
```





```
Frequency counts of corpus: corpus
bible        88
europarl     87
biomed       75
Name: count, dtype: int64

Counts of corpus in misclassified: corpus
bible        51
europarl     44
biomed       28
Name: count, dtype: int64
```

```
Counts of corpus in correctly classified: corpus
biomed      47
europarl    43
bible       37
Name: count, dtype: int64

Proportions of corpus in misclassified: corpus
bible       0.414634
europarl    0.357724
biomed      0.227642
Name: count, dtype: float64

Proportions of corpus in correctly classified: corpus
biomed      0.370079
europarl    0.338583
bible       0.291339
Name: count, dtype: float64
Subcorpus 'bible' symmetric KL divergence: 2.95432516672069e-06
Subcorpus 'biomed' symmetric KL divergence: 1.6558011492547724e-06
Subcorpus 'europarl' symmetric KL divergence: 3.947562264288915e-06
model results saved
```

Result

0.2.4 bert-large-cased

```
[ ]: # Define Experiment Parameters
     # named_model = "bert-base-cased"
     # named_model = "roberta-base"
     named_model = "bert-large-cased"
     # named_model = "roberta-large"
     # named_model = "" # modern bert
     ############
     regularization_weight_decay = 0.5
     learning_rate = 5e-6
     size_batch = 128
     length_max = 128
     num_epochs = 1
     ##########################################
     # x_col = "sentence"
     x_col = "sentence_no_contractions"
     # x_col = "pos_sequence"
     # x_col = "dep_sequence"
     # x_col = "morph_sequence"
     # x_col = "snc_pos_seq"
     # x_col = "snc_pos_alt"
     # x_col = "snc_morph_seq"
     # x_col = "snc_morph_alt"
     # x_col = "snc_dep_seq"
     # x_col = "snc_dep_alt"
     # x_col = "snc_morph_complexity_value"
     ############
     y_col = "binary_complexity"
     # y_col = "complexity"
     ############
     # x_task = "single"
     x_task = "multi"
     if x_task == "single":
         df_train = train_single_df
         df_val   = trial_val_single_df
         df_test  = test_single_df
     else:
         df_train = train_multi_df
         df_val   = trial_val_multi_df
         df_test  = test_multi_df
     ##########################################
     # Tokenize & Prepare Datasets
     train_data_hf = prepare_dataset(
         df_train,
         tokenizer,
         text_col=x_col,
         label_col=y_col,
         max_length=length_max)
     val_data_hf = prepare_dataset(
         df_val,
         tokenizer,
         text_col=x_col,
         label_col=y_col,
         max_length=length_max)
     test_data_hf = prepare_dataset(
         df_test,
         tokenizer,
         text_col=x_col,
         label_col=y_col,
         max_length=length_max)
     print("Datasets prepared. Sample from train_data_hf:\n", train_data_hf[10])
     # print("Datasets prepared. Sample from train_data_hf:\n", val_data_hf[10])
     # print("Datasets prepared. Sample from train_data_hf:\n", test_data_hf[10])
     ##########################################
     # custom_config = BertConfig.from_pretrained("roberta-base")
     # custom_config.hidden_act = "gelu"  # alts: "relu" "silu"
     # custom_config.attention_probs_dropout_prob = 0.1
     # custom_config.hidden_dropout_prob = 0.1
     # custom_config.gradient_checkpointing = False
```

```
############################################
model, tokenizer = get_model_and_tokenizer(
    remote_model_name="bert-large-cased",
    local_model_path=None,
    config=None)
############
# model, tokenizer = get_model_and_tokenizer(
#     remote_model_name=None
#     local_model_path="...CONFIGURE_PATH...",
#     config=custom_config)
print("==============")
print(named_model, ":")
print("==============")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters at load:", model.
  ↪num_parameters(only_trainable=True))
print("==============")
print("model lineage:", MODEL_LINEAGE)
print("==============")
```

```
Map:   0%|          | 0/1300 [00:00<?, ? examples/s]

Map:   0%|          | 0/250 [00:00<?, ? examples/s]

Map:   0%|          | 0/250 [00:00<?, ? examples/s]

Datasets prepared. Sample from train_data_hf:
 {'labels': tensor(0), 'input_ids': tensor([  101,  1109,  1210,  1808,  1104,
4836, 15949,   117,  1114,  1167,
         1190,   123,  1288,  8724,  1105, 24649,   117,  3657,  1107,  1103,
          163,  4626,  2758,  3293, 11171,   117,  1134, 11228, 11363,  1103,
         1352,  1104,  1103,  5655,  1206,  2470,  1386,  1105, 10585,   119,
          102,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0]),
'attention_mask': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0])}
Loading from Hugging Face model: bert-large-cased
```

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

```
=============
bert-large-cased :
=============
num_parameters: 333581314
num_trainable_parameters at load: 333581314
=============
model lineage: {'type': 'huggingface_hub', 'path': 'bert-large-cased',
'timestamp': '2025-04-14 01:54:01'}
=============
```

[ ]:
```
print(model)
```

```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(28996, 1024, padding_idx=0)
      (position_embeddings): Embedding(512, 1024)
      (token_type_embeddings): Embedding(2, 1024)
      (LayerNorm): LayerNorm((1024,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-23): 24 x BertLayer(
          (attention): BertAttention(
            (self): BertSdpaSelfAttention(
              (query): Linear(in_features=1024, out_features=1024, bias=True)
              (key): Linear(in_features=1024, out_features=1024, bias=True)
              (value): Linear(in_features=1024, out_features=1024, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=1024, out_features=1024, bias=True)
              (LayerNorm): LayerNorm((1024,), eps=1e-12,
elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=1024, out_features=4096, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=4096, out_features=1024, bias=True)
            (LayerNorm): LayerNorm((1024,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=1024, out_features=1024, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=1024, out_features=2, bias=True)
)
```

```
[ ]: for name, param in model.named_parameters():
         print(name, "requires_grad=", param.requires_grad)
```

```
bert.embeddings.word_embeddings.weight requires_grad= True
bert.embeddings.position_embeddings.weight requires_grad= True
bert.embeddings.token_type_embeddings.weight requires_grad= True
bert.embeddings.LayerNorm.weight requires_grad= True
bert.embeddings.LayerNorm.bias requires_grad= True
bert.encoder.layer.0.attention.self.query.weight requires_grad= True
bert.encoder.layer.0.attention.self.query.bias requires_grad= True
bert.encoder.layer.0.attention.self.key.weight requires_grad= True
bert.encoder.layer.0.attention.self.key.bias requires_grad= True
bert.encoder.layer.0.attention.self.value.weight requires_grad= True
bert.encoder.layer.0.attention.self.value.bias requires_grad= True
bert.encoder.layer.0.attention.output.dense.weight requires_grad= True
bert.encoder.layer.0.attention.output.dense.bias requires_grad= True
bert.encoder.layer.0.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.0.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.0.intermediate.dense.weight requires_grad= True
bert.encoder.layer.0.intermediate.dense.bias requires_grad= True
bert.encoder.layer.0.output.dense.weight requires_grad= True
bert.encoder.layer.0.output.dense.bias requires_grad= True
bert.encoder.layer.0.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.0.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.1.attention.self.query.weight requires_grad= True
bert.encoder.layer.1.attention.self.query.bias requires_grad= True
bert.encoder.layer.1.attention.self.key.weight requires_grad= True
bert.encoder.layer.1.attention.self.key.bias requires_grad= True
bert.encoder.layer.1.attention.self.value.weight requires_grad= True
bert.encoder.layer.1.attention.self.value.bias requires_grad= True
bert.encoder.layer.1.attention.output.dense.weight requires_grad= True
bert.encoder.layer.1.attention.output.dense.bias requires_grad= True
bert.encoder.layer.1.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.1.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.1.intermediate.dense.weight requires_grad= True
bert.encoder.layer.1.intermediate.dense.bias requires_grad= True
bert.encoder.layer.1.output.dense.weight requires_grad= True
bert.encoder.layer.1.output.dense.bias requires_grad= True
bert.encoder.layer.1.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.1.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.2.attention.self.query.weight requires_grad= True
bert.encoder.layer.2.attention.self.query.bias requires_grad= True
bert.encoder.layer.2.attention.self.key.weight requires_grad= True
bert.encoder.layer.2.attention.self.key.bias requires_grad= True
bert.encoder.layer.2.attention.self.value.weight requires_grad= True
bert.encoder.layer.2.attention.self.value.bias requires_grad= True
bert.encoder.layer.2.attention.output.dense.weight requires_grad= True
bert.encoder.layer.2.attention.output.dense.bias requires_grad= True
bert.encoder.layer.2.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.2.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.2.intermediate.dense.weight requires_grad= True
bert.encoder.layer.2.intermediate.dense.bias requires_grad= True
bert.encoder.layer.2.output.dense.weight requires_grad= True
bert.encoder.layer.2.output.dense.bias requires_grad= True
bert.encoder.layer.2.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.2.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.3.attention.self.query.weight requires_grad= True
bert.encoder.layer.3.attention.self.query.bias requires_grad= True
bert.encoder.layer.3.attention.self.key.weight requires_grad= True
bert.encoder.layer.3.attention.self.key.bias requires_grad= True
bert.encoder.layer.3.attention.self.value.weight requires_grad= True
bert.encoder.layer.3.attention.self.value.bias requires_grad= True
bert.encoder.layer.3.attention.output.dense.weight requires_grad= True
bert.encoder.layer.3.attention.output.dense.bias requires_grad= True
bert.encoder.layer.3.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.3.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.3.intermediate.dense.weight requires_grad= True
bert.encoder.layer.3.intermediate.dense.bias requires_grad= True
bert.encoder.layer.3.output.dense.weight requires_grad= True
bert.encoder.layer.3.output.dense.bias requires_grad= True
bert.encoder.layer.3.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.3.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.4.attention.self.query.weight requires_grad= True
bert.encoder.layer.4.attention.self.query.bias requires_grad= True
bert.encoder.layer.4.attention.self.key.weight requires_grad= True
bert.encoder.layer.4.attention.self.key.bias requires_grad= True
bert.encoder.layer.4.attention.self.value.weight requires_grad= True
bert.encoder.layer.4.attention.self.value.bias requires_grad= True
bert.encoder.layer.4.attention.output.dense.weight requires_grad= True
bert.encoder.layer.4.attention.output.dense.bias requires_grad= True
bert.encoder.layer.4.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.4.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.4.intermediate.dense.weight requires_grad= True
bert.encoder.layer.4.intermediate.dense.bias requires_grad= True
bert.encoder.layer.4.output.dense.weight requires_grad= True
bert.encoder.layer.4.output.dense.bias requires_grad= True
bert.encoder.layer.4.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.4.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.5.attention.self.query.weight requires_grad= True
bert.encoder.layer.5.attention.self.query.bias requires_grad= True
bert.encoder.layer.5.attention.self.key.weight requires_grad= True
bert.encoder.layer.5.attention.self.key.bias requires_grad= True
bert.encoder.layer.5.attention.self.value.weight requires_grad= True
bert.encoder.layer.5.attention.self.value.bias requires_grad= True
bert.encoder.layer.5.attention.output.dense.weight requires_grad= True
bert.encoder.layer.5.attention.output.dense.bias requires_grad= True
bert.encoder.layer.5.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.5.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.5.intermediate.dense.weight requires_grad= True
bert.encoder.layer.5.intermediate.dense.bias requires_grad= True
bert.encoder.layer.5.output.dense.weight requires_grad= True
bert.encoder.layer.5.output.dense.bias requires_grad= True
bert.encoder.layer.5.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.5.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.6.attention.self.query.weight requires_grad= True
bert.encoder.layer.6.attention.self.query.bias requires_grad= True
bert.encoder.layer.6.attention.self.key.weight requires_grad= True
bert.encoder.layer.6.attention.self.key.bias requires_grad= True
bert.encoder.layer.6.attention.self.value.weight requires_grad= True
bert.encoder.layer.6.attention.self.value.bias requires_grad= True
bert.encoder.layer.6.attention.output.dense.weight requires_grad= True
bert.encoder.layer.6.attention.output.dense.bias requires_grad= True
bert.encoder.layer.6.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.6.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.6.intermediate.dense.weight requires_grad= True
bert.encoder.layer.6.intermediate.dense.bias requires_grad= True
bert.encoder.layer.6.output.dense.weight requires_grad= True
bert.encoder.layer.6.output.dense.bias requires_grad= True
bert.encoder.layer.6.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.6.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.7.attention.self.query.weight requires_grad= True
bert.encoder.layer.7.attention.self.query.bias requires_grad= True
bert.encoder.layer.7.attention.self.key.weight requires_grad= True
bert.encoder.layer.7.attention.self.key.bias requires_grad= True
bert.encoder.layer.7.attention.self.value.weight requires_grad= True
bert.encoder.layer.7.attention.self.value.bias requires_grad= True
bert.encoder.layer.7.attention.output.dense.weight requires_grad= True
bert.encoder.layer.7.attention.output.dense.bias requires_grad= True
bert.encoder.layer.7.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.7.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.7.intermediate.dense.weight requires_grad= True
bert.encoder.layer.7.intermediate.dense.bias requires_grad= True
bert.encoder.layer.7.output.dense.weight requires_grad= True
bert.encoder.layer.7.output.dense.bias requires_grad= True
bert.encoder.layer.7.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.7.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.8.attention.self.query.weight requires_grad= True
bert.encoder.layer.8.attention.self.query.bias requires_grad= True
bert.encoder.layer.8.attention.self.key.weight requires_grad= True
bert.encoder.layer.8.attention.self.key.bias requires_grad= True
bert.encoder.layer.8.attention.self.value.weight requires_grad= True
bert.encoder.layer.8.attention.self.value.bias requires_grad= True
bert.encoder.layer.8.attention.output.dense.weight requires_grad= True
bert.encoder.layer.8.attention.output.dense.bias requires_grad= True
bert.encoder.layer.8.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.8.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.8.intermediate.dense.weight requires_grad= True
bert.encoder.layer.8.intermediate.dense.bias requires_grad= True
bert.encoder.layer.8.output.dense.weight requires_grad= True
bert.encoder.layer.8.output.dense.bias requires_grad= True
bert.encoder.layer.8.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.8.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.9.attention.self.query.weight requires_grad= True
bert.encoder.layer.9.attention.self.query.bias requires_grad= True
bert.encoder.layer.9.attention.self.key.weight requires_grad= True
bert.encoder.layer.9.attention.self.key.bias requires_grad= True
bert.encoder.layer.9.attention.self.value.weight requires_grad= True
bert.encoder.layer.9.attention.self.value.bias requires_grad= True
bert.encoder.layer.9.attention.output.dense.weight requires_grad= True
bert.encoder.layer.9.attention.output.dense.bias requires_grad= True
bert.encoder.layer.9.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.9.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.9.intermediate.dense.weight requires_grad= True
bert.encoder.layer.9.intermediate.dense.bias requires_grad= True
bert.encoder.layer.9.output.dense.weight requires_grad= True
bert.encoder.layer.9.output.dense.bias requires_grad= True
bert.encoder.layer.9.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.9.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.10.attention.self.query.weight requires_grad= True
bert.encoder.layer.10.attention.self.query.bias requires_grad= True
bert.encoder.layer.10.attention.self.key.weight requires_grad= True
bert.encoder.layer.10.attention.self.key.bias requires_grad= True
bert.encoder.layer.10.attention.self.value.weight requires_grad= True
bert.encoder.layer.10.attention.self.value.bias requires_grad= True
bert.encoder.layer.10.attention.output.dense.weight requires_grad= True
bert.encoder.layer.10.attention.output.dense.bias requires_grad= True
bert.encoder.layer.10.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.10.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.10.intermediate.dense.weight requires_grad= True
bert.encoder.layer.10.intermediate.dense.bias requires_grad= True
bert.encoder.layer.10.output.dense.weight requires_grad= True
bert.encoder.layer.10.output.dense.bias requires_grad= True
bert.encoder.layer.10.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.10.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.11.attention.self.query.weight requires_grad= True
bert.encoder.layer.11.attention.self.query.bias requires_grad= True
bert.encoder.layer.11.attention.self.key.weight requires_grad= True
bert.encoder.layer.11.attention.self.key.bias requires_grad= True
bert.encoder.layer.11.attention.self.value.weight requires_grad= True
bert.encoder.layer.11.attention.self.value.bias requires_grad= True
bert.encoder.layer.11.attention.output.dense.weight requires_grad= True
bert.encoder.layer.11.attention.output.dense.bias requires_grad= True
bert.encoder.layer.11.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.11.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.11.intermediate.dense.weight requires_grad= True
bert.encoder.layer.11.intermediate.dense.bias requires_grad= True
bert.encoder.layer.11.output.dense.weight requires_grad= True
bert.encoder.layer.11.output.dense.bias requires_grad= True
bert.encoder.layer.11.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.11.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.12.attention.self.query.weight requires_grad= True
bert.encoder.layer.12.attention.self.query.bias requires_grad= True
bert.encoder.layer.12.attention.self.key.weight requires_grad= True
bert.encoder.layer.12.attention.self.key.bias requires_grad= True
bert.encoder.layer.12.attention.self.value.weight requires_grad= True
bert.encoder.layer.12.attention.self.value.bias requires_grad= True
bert.encoder.layer.12.attention.output.dense.weight requires_grad= True
bert.encoder.layer.12.attention.output.dense.bias requires_grad= True
bert.encoder.layer.12.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.12.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.12.intermediate.dense.weight requires_grad= True
bert.encoder.layer.12.intermediate.dense.bias requires_grad= True
bert.encoder.layer.12.output.dense.weight requires_grad= True
bert.encoder.layer.12.output.dense.bias requires_grad= True
bert.encoder.layer.12.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.12.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.13.attention.self.query.weight requires_grad= True
bert.encoder.layer.13.attention.self.query.bias requires_grad= True
bert.encoder.layer.13.attention.self.key.weight requires_grad= True
bert.encoder.layer.13.attention.self.key.bias requires_grad= True
bert.encoder.layer.13.attention.self.value.weight requires_grad= True
bert.encoder.layer.13.attention.self.value.bias requires_grad= True
bert.encoder.layer.13.attention.output.dense.weight requires_grad= True
bert.encoder.layer.13.attention.output.dense.bias requires_grad= True
bert.encoder.layer.13.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.13.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.13.intermediate.dense.weight requires_grad= True
bert.encoder.layer.13.intermediate.dense.bias requires_grad= True
bert.encoder.layer.13.output.dense.weight requires_grad= True
bert.encoder.layer.13.output.dense.bias requires_grad= True
bert.encoder.layer.13.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.13.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.14.attention.self.query.weight requires_grad= True
bert.encoder.layer.14.attention.self.query.bias requires_grad= True
bert.encoder.layer.14.attention.self.key.weight requires_grad= True
bert.encoder.layer.14.attention.self.key.bias requires_grad= True
bert.encoder.layer.14.attention.self.value.weight requires_grad= True
bert.encoder.layer.14.attention.self.value.bias requires_grad= True
bert.encoder.layer.14.attention.output.dense.weight requires_grad= True
bert.encoder.layer.14.attention.output.dense.bias requires_grad= True
bert.encoder.layer.14.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.14.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.14.intermediate.dense.weight requires_grad= True
bert.encoder.layer.14.intermediate.dense.bias requires_grad= True
bert.encoder.layer.14.output.dense.weight requires_grad= True
bert.encoder.layer.14.output.dense.bias requires_grad= True
bert.encoder.layer.14.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.14.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.15.attention.self.query.weight requires_grad= True
bert.encoder.layer.15.attention.self.query.bias requires_grad= True
bert.encoder.layer.15.attention.self.key.weight requires_grad= True
bert.encoder.layer.15.attention.self.key.bias requires_grad= True
bert.encoder.layer.15.attention.self.value.weight requires_grad= True
bert.encoder.layer.15.attention.self.value.bias requires_grad= True
bert.encoder.layer.15.attention.output.dense.weight requires_grad= True
bert.encoder.layer.15.attention.output.dense.bias requires_grad= True
bert.encoder.layer.15.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.15.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.15.intermediate.dense.weight requires_grad= True
bert.encoder.layer.15.intermediate.dense.bias requires_grad= True
bert.encoder.layer.15.output.dense.weight requires_grad= True
bert.encoder.layer.15.output.dense.bias requires_grad= True
bert.encoder.layer.15.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.15.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.16.attention.self.query.weight requires_grad= True
bert.encoder.layer.16.attention.self.query.bias requires_grad= True
bert.encoder.layer.16.attention.self.key.weight requires_grad= True
bert.encoder.layer.16.attention.self.key.bias requires_grad= True
bert.encoder.layer.16.attention.self.value.weight requires_grad= True
bert.encoder.layer.16.attention.self.value.bias requires_grad= True
bert.encoder.layer.16.attention.output.dense.weight requires_grad= True
bert.encoder.layer.16.attention.output.dense.bias requires_grad= True
bert.encoder.layer.16.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.16.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.16.intermediate.dense.weight requires_grad= True
bert.encoder.layer.16.intermediate.dense.bias requires_grad= True
bert.encoder.layer.16.output.dense.weight requires_grad= True
bert.encoder.layer.16.output.dense.bias requires_grad= True
bert.encoder.layer.16.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.16.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.17.attention.self.query.weight requires_grad= True
bert.encoder.layer.17.attention.self.query.bias requires_grad= True
bert.encoder.layer.17.attention.self.key.weight requires_grad= True
bert.encoder.layer.17.attention.self.key.bias requires_grad= True
bert.encoder.layer.17.attention.self.value.weight requires_grad= True
bert.encoder.layer.17.attention.self.value.bias requires_grad= True
bert.encoder.layer.17.attention.output.dense.weight requires_grad= True
bert.encoder.layer.17.attention.output.dense.bias requires_grad= True
bert.encoder.layer.17.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.17.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.17.intermediate.dense.weight requires_grad= True
bert.encoder.layer.17.intermediate.dense.bias requires_grad= True
bert.encoder.layer.17.output.dense.weight requires_grad= True
bert.encoder.layer.17.output.dense.bias requires_grad= True
bert.encoder.layer.17.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.17.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.18.attention.self.query.weight requires_grad= True
bert.encoder.layer.18.attention.self.query.bias requires_grad= True
bert.encoder.layer.18.attention.self.key.weight requires_grad= True
bert.encoder.layer.18.attention.self.key.bias requires_grad= True
bert.encoder.layer.18.attention.self.value.weight requires_grad= True
bert.encoder.layer.18.attention.self.value.bias requires_grad= True
bert.encoder.layer.18.attention.output.dense.weight requires_grad= True
bert.encoder.layer.18.attention.output.dense.bias requires_grad= True
bert.encoder.layer.18.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.18.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.18.intermediate.dense.weight requires_grad= True
bert.encoder.layer.18.intermediate.dense.bias requires_grad= True
bert.encoder.layer.18.output.dense.weight requires_grad= True
bert.encoder.layer.18.output.dense.bias requires_grad= True
bert.encoder.layer.18.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.18.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.19.attention.self.query.weight requires_grad= True
bert.encoder.layer.19.attention.self.query.bias requires_grad= True
bert.encoder.layer.19.attention.self.key.weight requires_grad= True
bert.encoder.layer.19.attention.self.key.bias requires_grad= True
bert.encoder.layer.19.attention.self.value.weight requires_grad= True
bert.encoder.layer.19.attention.self.value.bias requires_grad= True
bert.encoder.layer.19.attention.output.dense.weight requires_grad= True
bert.encoder.layer.19.attention.output.dense.bias requires_grad= True
bert.encoder.layer.19.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.19.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.19.intermediate.dense.weight requires_grad= True
bert.encoder.layer.19.intermediate.dense.bias requires_grad= True
bert.encoder.layer.19.output.dense.weight requires_grad= True
bert.encoder.layer.19.output.dense.bias requires_grad= True
bert.encoder.layer.19.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.19.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.20.attention.self.query.weight requires_grad= True
bert.encoder.layer.20.attention.self.query.bias requires_grad= True
bert.encoder.layer.20.attention.self.key.weight requires_grad= True
bert.encoder.layer.20.attention.self.key.bias requires_grad= True
bert.encoder.layer.20.attention.self.value.weight requires_grad= True
bert.encoder.layer.20.attention.self.value.bias requires_grad= True
bert.encoder.layer.20.attention.output.dense.weight requires_grad= True
bert.encoder.layer.20.attention.output.dense.bias requires_grad= True
bert.encoder.layer.20.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.20.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.20.intermediate.dense.weight requires_grad= True
bert.encoder.layer.20.intermediate.dense.bias requires_grad= True
bert.encoder.layer.20.output.dense.weight requires_grad= True
bert.encoder.layer.20.output.dense.bias requires_grad= True
bert.encoder.layer.20.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.20.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.21.attention.self.query.weight requires_grad= True
bert.encoder.layer.21.attention.self.query.bias requires_grad= True
bert.encoder.layer.21.attention.self.key.weight requires_grad= True
bert.encoder.layer.21.attention.self.key.bias requires_grad= True
bert.encoder.layer.21.attention.self.value.weight requires_grad= True
bert.encoder.layer.21.attention.self.value.bias requires_grad= True
bert.encoder.layer.21.attention.output.dense.weight requires_grad= True
bert.encoder.layer.21.attention.output.dense.bias requires_grad= True
bert.encoder.layer.21.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.21.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.21.intermediate.dense.weight requires_grad= True
bert.encoder.layer.21.intermediate.dense.bias requires_grad= True
bert.encoder.layer.21.output.dense.weight requires_grad= True
bert.encoder.layer.21.output.dense.bias requires_grad= True
bert.encoder.layer.21.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.21.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.22.attention.self.query.weight requires_grad= True
bert.encoder.layer.22.attention.self.query.bias requires_grad= True
bert.encoder.layer.22.attention.self.key.weight requires_grad= True
bert.encoder.layer.22.attention.self.key.bias requires_grad= True
bert.encoder.layer.22.attention.self.value.weight requires_grad= True
bert.encoder.layer.22.attention.self.value.bias requires_grad= True
bert.encoder.layer.22.attention.output.dense.weight requires_grad= True
bert.encoder.layer.22.attention.output.dense.bias requires_grad= True
bert.encoder.layer.22.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.22.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.22.intermediate.dense.weight requires_grad= True
bert.encoder.layer.22.intermediate.dense.bias requires_grad= True
bert.encoder.layer.22.output.dense.weight requires_grad= True
bert.encoder.layer.22.output.dense.bias requires_grad= True
bert.encoder.layer.22.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.22.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.23.attention.self.query.weight requires_grad= True
bert.encoder.layer.23.attention.self.query.bias requires_grad= True
bert.encoder.layer.23.attention.self.key.weight requires_grad= True
bert.encoder.layer.23.attention.self.key.bias requires_grad= True
bert.encoder.layer.23.attention.self.value.weight requires_grad= True
bert.encoder.layer.23.attention.self.value.bias requires_grad= True
bert.encoder.layer.23.attention.output.dense.weight requires_grad= True
bert.encoder.layer.23.attention.output.dense.bias requires_grad= True
bert.encoder.layer.23.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.23.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.23.intermediate.dense.weight requires_grad= True
bert.encoder.layer.23.intermediate.dense.bias requires_grad= True
bert.encoder.layer.23.output.dense.weight requires_grad= True
bert.encoder.layer.23.output.dense.bias requires_grad= True
bert.encoder.layer.23.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.23.output.LayerNorm.bias requires_grad= True
bert.pooler.dense.weight requires_grad= True
bert.pooler.dense.bias requires_grad= True
classifier.weight requires_grad= True
classifier.bias requires_grad= True
```

[ ]:
```
##########################################
layers_to_unfreeze = [
    "bert.encoder.layer.23.",
    "bert.pooler.",
    "classifier.",
]
freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)
print(model.config)
print("=============")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
print("=============")
##########################################
print("Experiment configuration used with this experiment:")
print("model used:", named_model)
print("learning rate used:", learning_rate)
print("number of epochs:", num_epochs)
print("maximum sequence length:", length_max)
print("batch size used:", size_batch)
print("regularization value:", regularization_weight_decay)
print("outcome variable:", y_col)
print("task:", x_task)
print("input column:", x_col)
###########################################
print("=============")
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
```

```
BertConfig {
  "_attn_implementation_autoset": true,
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "directionality": "bidi",
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 1024,
  "initializer_range": 0.02,
  "intermediate_size": 4096,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 16,
  "num_hidden_layers": 24,
  "pad_token_id": 0,
  "pooler_fc_size": 768,
  "pooler_num_attention_heads": 12,
  "pooler_num_fc_layers": 3,
  "pooler_size_per_head": 128,
  "pooler_type": "first_token_transform",
  "position_embedding_type": "absolute",
  "torch_dtype": "float32",
  "transformers_version": "4.50.3",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 28996
}

=============
num_parameters: 333581314
num_trainable_parameters: 13647874
=============
Experiment configuration used with this experiment:
model used: bert-large-cased
```

```
learning rate used: 5e-06
number of epochs: 1
maximum sequence length: 128
batch size used: 128
regularization value: 0.5
outcome variable: binary_complexity
task: multi
input column: sentence_no_contractions
=============
num_trainable_parameters: 13647874
```

[ ]:
```
model.resize_token_embeddings(len(tokenizer))
```

[ ]:
```
Embedding(28996, 1024, padding_idx=0)
```

[ ]:
```
for name, param in model.named_parameters():
    print(name, "requires_grad=", param.requires_grad)
```

```
bert.embeddings.word_embeddings.weight requires_grad= False
bert.embeddings.position_embeddings.weight requires_grad= False
bert.embeddings.token_type_embeddings.weight requires_grad= False
bert.embeddings.LayerNorm.weight requires_grad= False
bert.embeddings.LayerNorm.bias requires_grad= False
bert.encoder.layer.0.attention.self.query.weight requires_grad= False
bert.encoder.layer.0.attention.self.query.bias requires_grad= False
bert.encoder.layer.0.attention.self.key.weight requires_grad= False
bert.encoder.layer.0.attention.self.key.bias requires_grad= False
bert.encoder.layer.0.attention.self.value.weight requires_grad= False
bert.encoder.layer.0.attention.self.value.bias requires_grad= False
bert.encoder.layer.0.attention.output.dense.weight requires_grad= False
bert.encoder.layer.0.attention.output.dense.bias requires_grad= False
bert.encoder.layer.0.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.0.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.0.intermediate.dense.weight requires_grad= False
bert.encoder.layer.0.intermediate.dense.bias requires_grad= False
bert.encoder.layer.0.output.dense.weight requires_grad= False
bert.encoder.layer.0.output.dense.bias requires_grad= False
bert.encoder.layer.0.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.0.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.1.attention.self.query.weight requires_grad= False
bert.encoder.layer.1.attention.self.query.bias requires_grad= False
bert.encoder.layer.1.attention.self.key.weight requires_grad= False
bert.encoder.layer.1.attention.self.key.bias requires_grad= False
bert.encoder.layer.1.attention.self.value.weight requires_grad= False
bert.encoder.layer.1.attention.self.value.bias requires_grad= False
bert.encoder.layer.1.attention.output.dense.weight requires_grad= False
bert.encoder.layer.1.attention.output.dense.bias requires_grad= False
bert.encoder.layer.1.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.1.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.1.intermediate.dense.weight requires_grad= False
bert.encoder.layer.1.intermediate.dense.bias requires_grad= False
bert.encoder.layer.1.output.dense.weight requires_grad= False
bert.encoder.layer.1.output.dense.bias requires_grad= False
bert.encoder.layer.1.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.1.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.2.attention.self.query.weight requires_grad= False
bert.encoder.layer.2.attention.self.query.bias requires_grad= False
bert.encoder.layer.2.attention.self.key.weight requires_grad= False
bert.encoder.layer.2.attention.self.key.bias requires_grad= False
bert.encoder.layer.2.attention.self.value.weight requires_grad= False
bert.encoder.layer.2.attention.self.value.bias requires_grad= False
bert.encoder.layer.2.attention.output.dense.weight requires_grad= False
bert.encoder.layer.2.attention.output.dense.bias requires_grad= False
bert.encoder.layer.2.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.2.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.2.intermediate.dense.weight requires_grad= False
bert.encoder.layer.2.intermediate.dense.bias requires_grad= False
bert.encoder.layer.2.output.dense.weight requires_grad= False
bert.encoder.layer.2.output.dense.bias requires_grad= False
bert.encoder.layer.2.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.2.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.3.attention.self.query.weight requires_grad= False
bert.encoder.layer.3.attention.self.query.bias requires_grad= False
bert.encoder.layer.3.attention.self.key.weight requires_grad= False
bert.encoder.layer.3.attention.self.key.bias requires_grad= False
bert.encoder.layer.3.attention.self.value.weight requires_grad= False
bert.encoder.layer.3.attention.self.value.bias requires_grad= False
bert.encoder.layer.3.attention.output.dense.weight requires_grad= False
bert.encoder.layer.3.attention.output.dense.bias requires_grad= False
bert.encoder.layer.3.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.3.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.3.intermediate.dense.weight requires_grad= False
bert.encoder.layer.3.intermediate.dense.bias requires_grad= False
bert.encoder.layer.3.output.dense.weight requires_grad= False
bert.encoder.layer.3.output.dense.bias requires_grad= False
bert.encoder.layer.3.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.3.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.4.attention.self.query.weight requires_grad= False
bert.encoder.layer.4.attention.self.query.bias requires_grad= False
bert.encoder.layer.4.attention.self.key.weight requires_grad= False
bert.encoder.layer.4.attention.self.key.bias requires_grad= False
bert.encoder.layer.4.attention.self.value.weight requires_grad= False
bert.encoder.layer.4.attention.self.value.bias requires_grad= False
bert.encoder.layer.4.attention.output.dense.weight requires_grad= False
bert.encoder.layer.4.attention.output.dense.bias requires_grad= False
bert.encoder.layer.4.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.4.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.4.intermediate.dense.weight requires_grad= False
bert.encoder.layer.4.intermediate.dense.bias requires_grad= False
bert.encoder.layer.4.output.dense.weight requires_grad= False
bert.encoder.layer.4.output.dense.bias requires_grad= False
bert.encoder.layer.4.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.4.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.5.attention.self.query.weight requires_grad= False
bert.encoder.layer.5.attention.self.query.bias requires_grad= False
bert.encoder.layer.5.attention.self.key.weight requires_grad= False
bert.encoder.layer.5.attention.self.key.bias requires_grad= False
bert.encoder.layer.5.attention.self.value.weight requires_grad= False
bert.encoder.layer.5.attention.self.value.bias requires_grad= False
bert.encoder.layer.5.attention.output.dense.weight requires_grad= False
bert.encoder.layer.5.attention.output.dense.bias requires_grad= False
bert.encoder.layer.5.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.5.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.5.intermediate.dense.weight requires_grad= False
bert.encoder.layer.5.intermediate.dense.bias requires_grad= False
bert.encoder.layer.5.output.dense.weight requires_grad= False
bert.encoder.layer.5.output.dense.bias requires_grad= False
bert.encoder.layer.5.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.5.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.6.attention.self.query.weight requires_grad= False
bert.encoder.layer.6.attention.self.query.bias requires_grad= False
bert.encoder.layer.6.attention.self.key.weight requires_grad= False
bert.encoder.layer.6.attention.self.key.bias requires_grad= False
bert.encoder.layer.6.attention.self.value.weight requires_grad= False
bert.encoder.layer.6.attention.self.value.bias requires_grad= False
bert.encoder.layer.6.attention.output.dense.weight requires_grad= False
bert.encoder.layer.6.attention.output.dense.bias requires_grad= False
bert.encoder.layer.6.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.6.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.6.intermediate.dense.weight requires_grad= False
bert.encoder.layer.6.intermediate.dense.bias requires_grad= False
bert.encoder.layer.6.output.dense.weight requires_grad= False
bert.encoder.layer.6.output.dense.bias requires_grad= False
bert.encoder.layer.6.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.6.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.7.attention.self.query.weight requires_grad= False
bert.encoder.layer.7.attention.self.query.bias requires_grad= False
bert.encoder.layer.7.attention.self.key.weight requires_grad= False
bert.encoder.layer.7.attention.self.key.bias requires_grad= False
bert.encoder.layer.7.attention.self.value.weight requires_grad= False
bert.encoder.layer.7.attention.self.value.bias requires_grad= False
bert.encoder.layer.7.attention.output.dense.weight requires_grad= False
bert.encoder.layer.7.attention.output.dense.bias requires_grad= False
bert.encoder.layer.7.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.7.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.7.intermediate.dense.weight requires_grad= False
bert.encoder.layer.7.intermediate.dense.bias requires_grad= False
bert.encoder.layer.7.output.dense.weight requires_grad= False
bert.encoder.layer.7.output.dense.bias requires_grad= False
bert.encoder.layer.7.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.7.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.8.attention.self.query.weight requires_grad= False
bert.encoder.layer.8.attention.self.query.bias requires_grad= False
bert.encoder.layer.8.attention.self.key.weight requires_grad= False
bert.encoder.layer.8.attention.self.key.bias requires_grad= False
bert.encoder.layer.8.attention.self.value.weight requires_grad= False
bert.encoder.layer.8.attention.self.value.bias requires_grad= False
bert.encoder.layer.8.attention.output.dense.weight requires_grad= False
bert.encoder.layer.8.attention.output.dense.bias requires_grad= False
bert.encoder.layer.8.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.8.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.8.intermediate.dense.weight requires_grad= False
bert.encoder.layer.8.intermediate.dense.bias requires_grad= False
bert.encoder.layer.8.output.dense.weight requires_grad= False
bert.encoder.layer.8.output.dense.bias requires_grad= False
bert.encoder.layer.8.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.8.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.9.attention.self.query.weight requires_grad= False
bert.encoder.layer.9.attention.self.query.bias requires_grad= False
bert.encoder.layer.9.attention.self.key.weight requires_grad= False
bert.encoder.layer.9.attention.self.key.bias requires_grad= False
bert.encoder.layer.9.attention.self.value.weight requires_grad= False
bert.encoder.layer.9.attention.self.value.bias requires_grad= False
bert.encoder.layer.9.attention.output.dense.weight requires_grad= False
bert.encoder.layer.9.attention.output.dense.bias requires_grad= False
bert.encoder.layer.9.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.9.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.9.intermediate.dense.weight requires_grad= False
bert.encoder.layer.9.intermediate.dense.bias requires_grad= False
bert.encoder.layer.9.output.dense.weight requires_grad= False
bert.encoder.layer.9.output.dense.bias requires_grad= False
bert.encoder.layer.9.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.9.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.10.attention.self.query.weight requires_grad= False
bert.encoder.layer.10.attention.self.query.bias requires_grad= False
bert.encoder.layer.10.attention.self.key.weight requires_grad= False
bert.encoder.layer.10.attention.self.key.bias requires_grad= False
bert.encoder.layer.10.attention.self.value.weight requires_grad= False
bert.encoder.layer.10.attention.self.value.bias requires_grad= False
bert.encoder.layer.10.attention.output.dense.weight requires_grad= False
bert.encoder.layer.10.attention.output.dense.bias requires_grad= False
bert.encoder.layer.10.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.10.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.10.intermediate.dense.weight requires_grad= False
bert.encoder.layer.10.intermediate.dense.bias requires_grad= False
bert.encoder.layer.10.output.dense.weight requires_grad= False
bert.encoder.layer.10.output.dense.bias requires_grad= False
bert.encoder.layer.10.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.10.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.11.attention.self.query.weight requires_grad= False
bert.encoder.layer.11.attention.self.query.bias requires_grad= False
bert.encoder.layer.11.attention.self.key.weight requires_grad= False
bert.encoder.layer.11.attention.self.key.bias requires_grad= False
bert.encoder.layer.11.attention.self.value.weight requires_grad= False
bert.encoder.layer.11.attention.self.value.bias requires_grad= False
bert.encoder.layer.11.attention.output.dense.weight requires_grad= False
bert.encoder.layer.11.attention.output.dense.bias requires_grad= False
bert.encoder.layer.11.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.11.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.11.intermediate.dense.weight requires_grad= False
bert.encoder.layer.11.intermediate.dense.bias requires_grad= False
bert.encoder.layer.11.output.dense.weight requires_grad= False
bert.encoder.layer.11.output.dense.bias requires_grad= False
bert.encoder.layer.11.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.11.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.12.attention.self.query.weight requires_grad= False
bert.encoder.layer.12.attention.self.query.bias requires_grad= False
bert.encoder.layer.12.attention.self.key.weight requires_grad= False
bert.encoder.layer.12.attention.self.key.bias requires_grad= False
bert.encoder.layer.12.attention.self.value.weight requires_grad= False
bert.encoder.layer.12.attention.self.value.bias requires_grad= False
bert.encoder.layer.12.attention.output.dense.weight requires_grad= False
bert.encoder.layer.12.attention.output.dense.bias requires_grad= False
bert.encoder.layer.12.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.12.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.12.intermediate.dense.weight requires_grad= False
bert.encoder.layer.12.intermediate.dense.bias requires_grad= False
bert.encoder.layer.12.output.dense.weight requires_grad= False
bert.encoder.layer.12.output.dense.bias requires_grad= False
bert.encoder.layer.12.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.12.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.13.attention.self.query.weight requires_grad= False
bert.encoder.layer.13.attention.self.query.bias requires_grad= False
bert.encoder.layer.13.attention.self.key.weight requires_grad= False
bert.encoder.layer.13.attention.self.key.bias requires_grad= False
bert.encoder.layer.13.attention.self.value.weight requires_grad= False
bert.encoder.layer.13.attention.self.value.bias requires_grad= False
bert.encoder.layer.13.attention.output.dense.weight requires_grad= False
bert.encoder.layer.13.attention.output.dense.bias requires_grad= False
bert.encoder.layer.13.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.13.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.13.intermediate.dense.weight requires_grad= False
bert.encoder.layer.13.intermediate.dense.bias requires_grad= False
bert.encoder.layer.13.output.dense.weight requires_grad= False
bert.encoder.layer.13.output.dense.bias requires_grad= False
bert.encoder.layer.13.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.13.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.14.attention.self.query.weight requires_grad= False
bert.encoder.layer.14.attention.self.query.bias requires_grad= False
bert.encoder.layer.14.attention.self.key.weight requires_grad= False
bert.encoder.layer.14.attention.self.key.bias requires_grad= False
bert.encoder.layer.14.attention.self.value.weight requires_grad= False
bert.encoder.layer.14.attention.self.value.bias requires_grad= False
bert.encoder.layer.14.attention.output.dense.weight requires_grad= False
bert.encoder.layer.14.attention.output.dense.bias requires_grad= False
bert.encoder.layer.14.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.14.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.14.intermediate.dense.weight requires_grad= False
bert.encoder.layer.14.intermediate.dense.bias requires_grad= False
bert.encoder.layer.14.output.dense.weight requires_grad= False
bert.encoder.layer.14.output.dense.bias requires_grad= False
bert.encoder.layer.14.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.14.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.15.attention.self.query.weight requires_grad= False
bert.encoder.layer.15.attention.self.query.bias requires_grad= False
bert.encoder.layer.15.attention.self.key.weight requires_grad= False
bert.encoder.layer.15.attention.self.key.bias requires_grad= False
bert.encoder.layer.15.attention.self.value.weight requires_grad= False
bert.encoder.layer.15.attention.self.value.bias requires_grad= False
bert.encoder.layer.15.attention.output.dense.weight requires_grad= False
bert.encoder.layer.15.attention.output.dense.bias requires_grad= False
bert.encoder.layer.15.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.15.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.15.intermediate.dense.weight requires_grad= False
bert.encoder.layer.15.intermediate.dense.bias requires_grad= False
bert.encoder.layer.15.output.dense.weight requires_grad= False
bert.encoder.layer.15.output.dense.bias requires_grad= False
bert.encoder.layer.15.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.15.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.16.attention.self.query.weight requires_grad= False
bert.encoder.layer.16.attention.self.query.bias requires_grad= False
bert.encoder.layer.16.attention.self.key.weight requires_grad= False
bert.encoder.layer.16.attention.self.key.bias requires_grad= False
bert.encoder.layer.16.attention.self.value.weight requires_grad= False
bert.encoder.layer.16.attention.self.value.bias requires_grad= False
bert.encoder.layer.16.attention.output.dense.weight requires_grad= False
bert.encoder.layer.16.attention.output.dense.bias requires_grad= False
bert.encoder.layer.16.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.16.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.16.intermediate.dense.weight requires_grad= False
bert.encoder.layer.16.intermediate.dense.bias requires_grad= False
bert.encoder.layer.16.output.dense.weight requires_grad= False
bert.encoder.layer.16.output.dense.bias requires_grad= False
bert.encoder.layer.16.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.16.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.17.attention.self.query.weight requires_grad= False
bert.encoder.layer.17.attention.self.query.bias requires_grad= False
bert.encoder.layer.17.attention.self.key.weight requires_grad= False
bert.encoder.layer.17.attention.self.key.bias requires_grad= False
bert.encoder.layer.17.attention.self.value.weight requires_grad= False
bert.encoder.layer.17.attention.self.value.bias requires_grad= False
bert.encoder.layer.17.attention.output.dense.weight requires_grad= False
bert.encoder.layer.17.attention.output.dense.bias requires_grad= False
bert.encoder.layer.17.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.17.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.17.intermediate.dense.weight requires_grad= False
bert.encoder.layer.17.intermediate.dense.bias requires_grad= False
bert.encoder.layer.17.output.dense.weight requires_grad= False
bert.encoder.layer.17.output.dense.bias requires_grad= False
bert.encoder.layer.17.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.17.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.18.attention.self.query.weight requires_grad= False
bert.encoder.layer.18.attention.self.query.bias requires_grad= False
bert.encoder.layer.18.attention.self.key.weight requires_grad= False
bert.encoder.layer.18.attention.self.key.bias requires_grad= False
bert.encoder.layer.18.attention.self.value.weight requires_grad= False
bert.encoder.layer.18.attention.self.value.bias requires_grad= False
bert.encoder.layer.18.attention.output.dense.weight requires_grad= False
bert.encoder.layer.18.attention.output.dense.bias requires_grad= False
bert.encoder.layer.18.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.18.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.18.intermediate.dense.weight requires_grad= False
bert.encoder.layer.18.intermediate.dense.bias requires_grad= False
bert.encoder.layer.18.output.dense.weight requires_grad= False
bert.encoder.layer.18.output.dense.bias requires_grad= False
bert.encoder.layer.18.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.18.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.19.attention.self.query.weight requires_grad= False
bert.encoder.layer.19.attention.self.query.bias requires_grad= False
bert.encoder.layer.19.attention.self.key.weight requires_grad= False
bert.encoder.layer.19.attention.self.key.bias requires_grad= False
bert.encoder.layer.19.attention.self.value.weight requires_grad= False
bert.encoder.layer.19.attention.self.value.bias requires_grad= False
bert.encoder.layer.19.attention.output.dense.weight requires_grad= False
bert.encoder.layer.19.attention.output.dense.bias requires_grad= False
bert.encoder.layer.19.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.19.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.19.intermediate.dense.weight requires_grad= False
bert.encoder.layer.19.intermediate.dense.bias requires_grad= False
bert.encoder.layer.19.output.dense.weight requires_grad= False
bert.encoder.layer.19.output.dense.bias requires_grad= False
bert.encoder.layer.19.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.19.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.20.attention.self.query.weight requires_grad= False
bert.encoder.layer.20.attention.self.query.bias requires_grad= False
bert.encoder.layer.20.attention.self.key.weight requires_grad= False
bert.encoder.layer.20.attention.self.key.bias requires_grad= False
bert.encoder.layer.20.attention.self.value.weight requires_grad= False
bert.encoder.layer.20.attention.self.value.bias requires_grad= False
bert.encoder.layer.20.attention.output.dense.weight requires_grad= False
bert.encoder.layer.20.attention.output.dense.bias requires_grad= False
bert.encoder.layer.20.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.20.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.20.intermediate.dense.weight requires_grad= False
bert.encoder.layer.20.intermediate.dense.bias requires_grad= False
bert.encoder.layer.20.output.dense.weight requires_grad= False
bert.encoder.layer.20.output.dense.bias requires_grad= False
bert.encoder.layer.20.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.20.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.21.attention.self.query.weight requires_grad= False
bert.encoder.layer.21.attention.self.query.bias requires_grad= False
bert.encoder.layer.21.attention.self.key.weight requires_grad= False
bert.encoder.layer.21.attention.self.key.bias requires_grad= False
bert.encoder.layer.21.attention.self.value.weight requires_grad= False
bert.encoder.layer.21.attention.self.value.bias requires_grad= False
bert.encoder.layer.21.attention.output.dense.weight requires_grad= False
bert.encoder.layer.21.attention.output.dense.bias requires_grad= False
bert.encoder.layer.21.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.21.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.21.intermediate.dense.weight requires_grad= False
bert.encoder.layer.21.intermediate.dense.bias requires_grad= False
bert.encoder.layer.21.output.dense.weight requires_grad= False
bert.encoder.layer.21.output.dense.bias requires_grad= False
bert.encoder.layer.21.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.21.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.22.attention.self.query.weight requires_grad= False
bert.encoder.layer.22.attention.self.query.bias requires_grad= False
bert.encoder.layer.22.attention.self.key.weight requires_grad= False
bert.encoder.layer.22.attention.self.key.bias requires_grad= False
bert.encoder.layer.22.attention.self.value.weight requires_grad= False
bert.encoder.layer.22.attention.self.value.bias requires_grad= False
bert.encoder.layer.22.attention.output.dense.weight requires_grad= False
bert.encoder.layer.22.attention.output.dense.bias requires_grad= False
bert.encoder.layer.22.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.22.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.22.intermediate.dense.weight requires_grad= False
bert.encoder.layer.22.intermediate.dense.bias requires_grad= False
bert.encoder.layer.22.output.dense.weight requires_grad= False
bert.encoder.layer.22.output.dense.bias requires_grad= False
bert.encoder.layer.22.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.22.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.23.attention.self.query.weight requires_grad= True
bert.encoder.layer.23.attention.self.query.bias requires_grad= True
bert.encoder.layer.23.attention.self.key.weight requires_grad= True
bert.encoder.layer.23.attention.self.key.bias requires_grad= True
bert.encoder.layer.23.attention.self.value.weight requires_grad= True
bert.encoder.layer.23.attention.self.value.bias requires_grad= True
bert.encoder.layer.23.attention.output.dense.weight requires_grad= True
bert.encoder.layer.23.attention.output.dense.bias requires_grad= True
bert.encoder.layer.23.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.23.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.23.intermediate.dense.weight requires_grad= True
bert.encoder.layer.23.intermediate.dense.bias requires_grad= True
bert.encoder.layer.23.output.dense.weight requires_grad= True
bert.encoder.layer.23.output.dense.bias requires_grad= True
bert.encoder.layer.23.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.23.output.LayerNorm.bias requires_grad= True
bert.pooler.dense.weight requires_grad= True
bert.pooler.dense.bias requires_grad= True
classifier.weight requires_grad= True
classifier.bias requires_grad= True
```

[ ]:
```
model.resize_token_embeddings(len(tokenizer))
```

[ ]:
```
Embedding(28996, 1024, padding_idx=0)
```

[ ]:
```
# Train & Evaluate
trained_model, trainer_obj = train_transformer_model(
    model = model,
    tokenizer = tokenizer,
    train_dataset = train_data_hf,
    val_dataset = val_data_hf,
    output_dir = dir_results,
    num_epochs = num_epochs,
    batch_size = size_batch,
    lr = learning_rate,
    weight_decay = regularization_weight_decay)
metrics = trainer_obj.evaluate()
print("Validation metrics:", metrics)
test_metrics = trainer_obj.evaluate(test_data_hf) if test_data_hf else None
print("Test metrics:", test_metrics)
```

```
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1611:
FutureWarning: `evaluation_strategy` is deprecated and will be removed in
version 4.46 of  Transformers. Use `eval_strategy` instead
  warnings.warn(
<ipython-input-20-81222a87c90b>:31: FutureWarning: `tokenizer` is deprecated and
will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class`
instead.
  trainer = Trainer(
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
Validation metrics: {'eval_loss': 0.7082364559173584, 'eval_accuracy': 0.572,
'eval_precision': 0.5106382978723404, 'eval_recall': 0.2222222222222222,
'eval_f1': 0.3096774193548387, 'eval_runtime': 2.2694,
'eval_samples_per_second': 110.16, 'eval_steps_per_second': 0.881, 'epoch': 1.0}
Test metrics: {'eval_loss': 0.7516778707504272, 'eval_accuracy': 0.5,
'eval_precision': 0.5217391304347826, 'eval_recall': 0.1889763779527559,
'eval_f1': 0.2774566473988439, 'eval_runtime': 2.4586,
'eval_samples_per_second': 101.685, 'eval_steps_per_second': 0.813, 'epoch':
1.0}
```

[ ]:
```
# save model checkpoint
# timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
pacific_time = datetime.now(zoneinfo.ZoneInfo("America/Los_Angeles"))
timestamp = pacific_time.isoformat()
model_save_path = os.path.join(dir_models,␣
  ↪f"{x_task}_{named_model}_{y_col}_{timestamp}")
trainer_obj.save_model(model_save_path)
print(f"Model checkpoint saved to: {model_save_path}")
# log experiment results
experiment_info = {
    "model_name": named_model,
    "learning_rate": learning_rate,
    "epochs": num_epochs,
    "batch_size": size_batch,
    "weight_decay": regularization_weight_decay,
    "x_task": x_task,
    "x_col": x_col,
    "y_col": y_col,
    "layers_to_unfreeze": layers_to_unfreeze}
model_info = gather_model_details(trained_model)
all_run_metrics = gather_all_run_metrics(
    trainer=trainer_obj,
    train_dataset=train_data_hf,
    val_dataset=val_data_hf,
    test_dataset=test_data_hf)
log_experiment_results_json(
    experiment_meta=experiment_info,
    model_details=model_info,
    run_metrics=all_run_metrics,
    log_file=log_filepath)
print(f"EXPERIMENT LOGGED TO: {log_filepath}")
```

```
Model checkpoint saved to: /content/drive/MyDrive/266-final/models/multi_bert-
large-cased_binary_complexity_2025-04-13T18:54:16.408669-07:00

<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:
/content/drive/MyDrive/266-final/results/experiment_runs.txt
```

[ ]:
```
prediction_output = trainer_obj.predict(test_data_hf)
raw_predictions = prediction_output.predictions
true_labels = prediction_output.label_ids
preds = np.argmax(raw_predictions, axis=1)
mismatch_indices = np.where(preds != true_labels)[0]

error_rows = []
for idx in mismatch_indices:
    text_value = df_test.iloc[idx][x_col]
    true_label_val = true_labels[idx]
    pred_label_val = preds[idx]

    error_rows.append({
        "hf_index": idx,
        "text": text_value,
        "true_label": true_label_val,
        "predicted_label": pred_label_val
    })

error_df = pd.DataFrame(error_rows)
df_test_for_merge = df_test.copy()
df_test_for_merge["error_matching_prefix"] = df_test_for_merge[x_col].str[:50]
# df_test_for_merge.drop(columns=[x_col], inplace=True)

error_df["error_matching_prefix"] = error_df["text"].str[:50]
error_df = error_df.merge(
    df_test_for_merge,
    on="error_matching_prefix",
    how="left",
    suffixes=("", "_source"))

error_df.to_csv("bert-base-cased_mismatches.csv", index=False)

# print("Number of misclassified samples:", len(error_df))
```

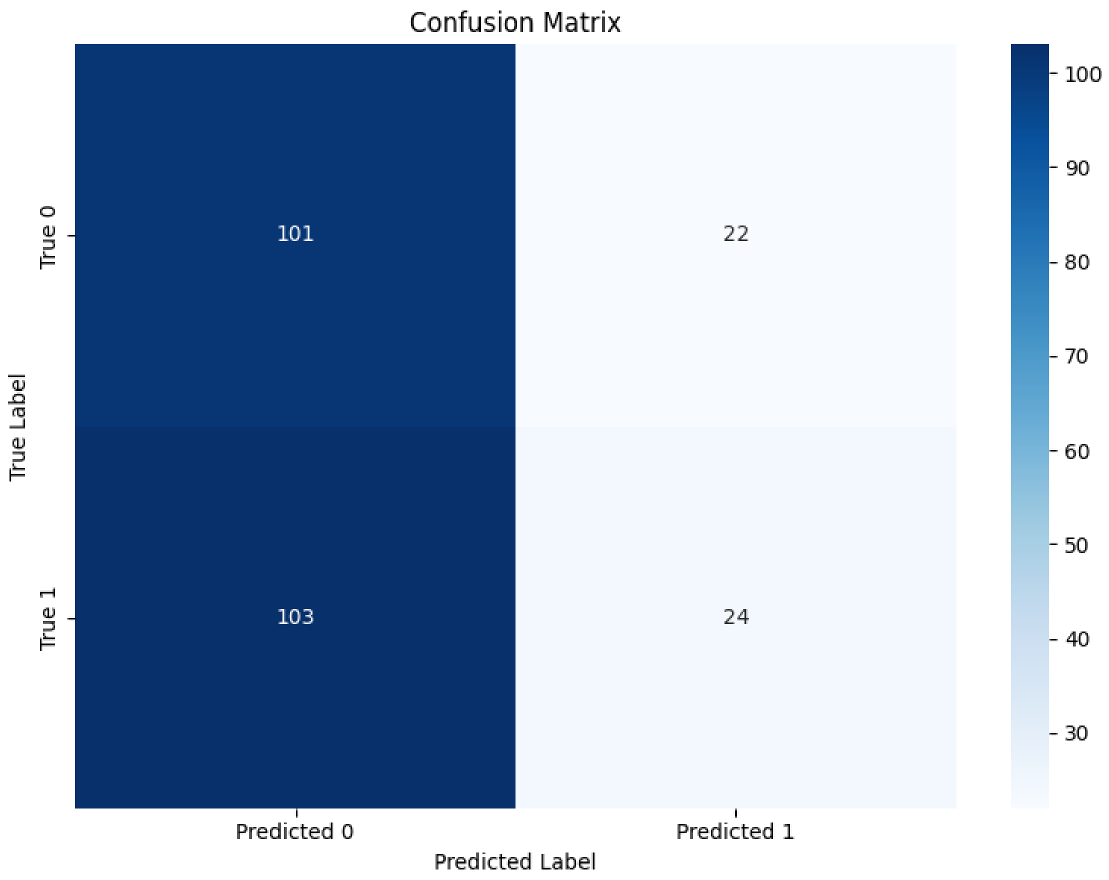
```
print("\nMerged error_df with extra columns:")
# display(error_df.head(15))

# print("\nConfusion Matrix:")
# cm = confusion_matrix(true_labels, preds)
# plt.figure(figsize=(8, 6))
# sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True,
#             xticklabels=["Predicted 0", "Predicted 1"],
#             yticklabels=["True 0", "True 1"])
# plt.xlabel('Predicted Label')
# plt.ylabel('True Label')
# plt.title('Confusion Matrix')
# plt.tight_layout()
# plt.show()
# print("confusion matrix metrics: \n", cm)
# error_save_path = os.path.join(dir_results,␣
 ↪f"{x_task}_{named_model}_{y_col}_{x_col}_errors.csv")
# error_df.to_csv(error_save_path, index=False)
# print("Result saved to results directory.")
prediction_output = trainer_obj.predict(test_data_hf)
raw_predictions = prediction_output.predictions
true_labels = prediction_output.label_ids
preds = np.argmax(raw_predictions, axis=1)
df_test["true_label"] = true_labels
df_test["predicted_label"] = preds
df_test["is_incorrect"] = (df_test["predicted_label"] != df_test["true_label"])
df_test["avg_embedding"] = None
device = next(model.parameters()).device
for i in range(len(df_test)):
    text_value = df_test.iloc[i][x_col]
    e = tokenizer(text_value, return_tensors="pt", truncation=True,␣
 ↪max_length=512).to(device)
    with torch.no_grad():
        emb = model.bert.embeddings.word_embeddings(e["input_ids"]).mean(dim=1).
 ↪squeeze().cpu().numpy()
    df_test.at[i,"avg_embedding"] = emb
cm = confusion_matrix(df_test["true_label"], df_test["predicted_label"])
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True,␣
 ↪xticklabels=["Predicted 0","Predicted 1"], yticklabels=["True 0","True 1"])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.tight_layout()
plt.show()
# print("confusion matrix metrics:\n", cm)
```

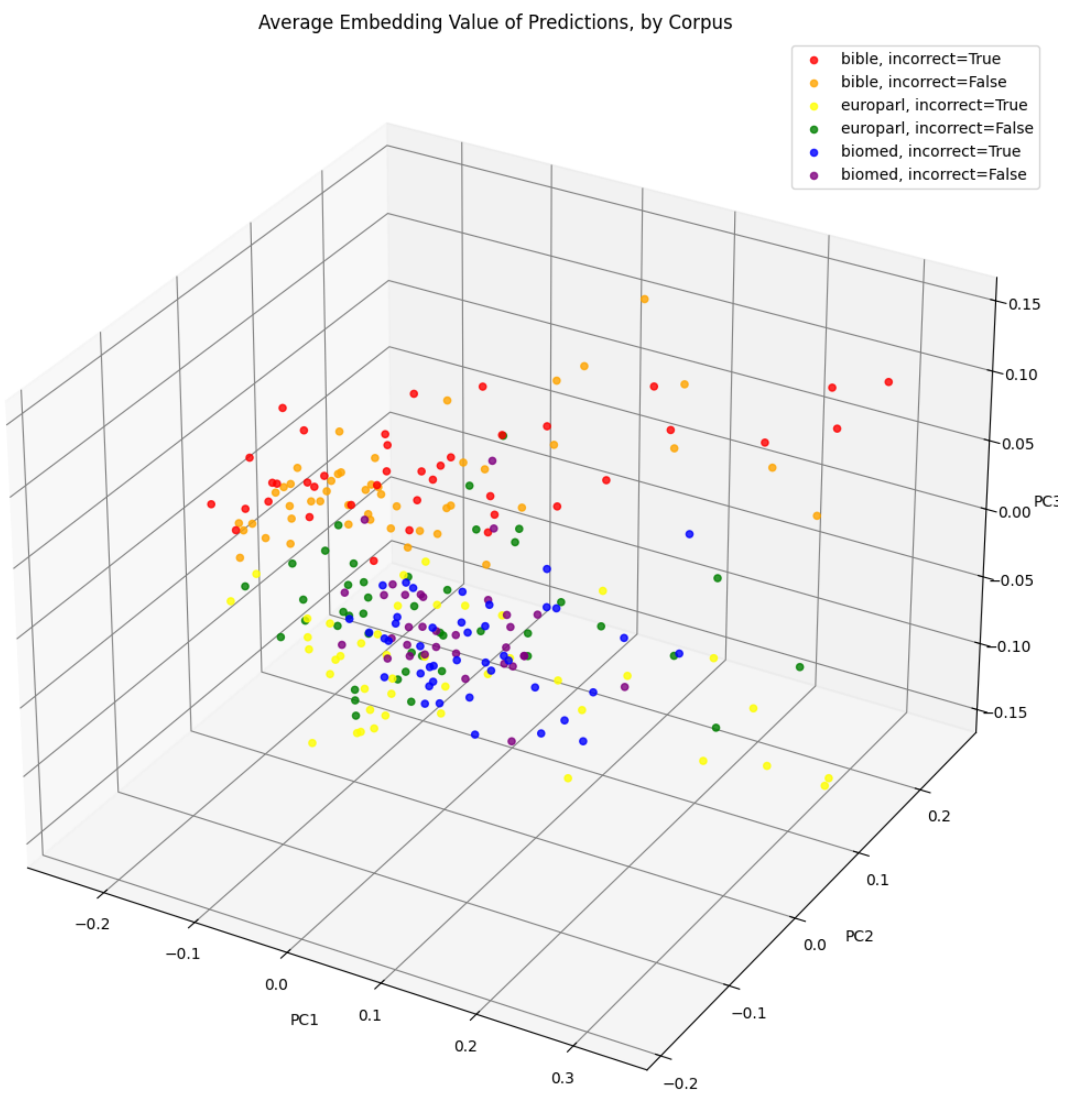
```
df_plot = df_test.dropna(subset=["avg_embedding"]).copy()
embeddings = np.stack(df_plot["avg_embedding"].values)
pca = PCA(n_components=3)
reduced = pca.fit_transform(embeddings)
df_plot["pca_x"] = reduced[:,0]
df_plot["pca_y"] = reduced[:,1]
df_plot["pca_z"] = reduced[:,2]
colors = {
    ("bible", True): "red",
    ("bible", False): "orange",
    ("europarl", True): "yellow",
    ("europarl", False): "green",
    ("biomed", True): "blue",
    ("biomed", False): "purple"}
fig = plt.figure(figsize=(13,13))
ax = fig.add_subplot(111, projection='3d')
fig.set_facecolor("white")
ax.set_facecolor("white")
ax.xaxis._axinfo["grid"]["color"] = "gray"
ax.yaxis._axinfo["grid"]["color"] = "gray"
ax.zaxis._axinfo["grid"]["color"] = "gray"
for (corp, incorr), color in colors.items():
    subset = df_plot[(df_plot["corpus"]==corp) &␣
  ↪(df_plot["is_incorrect"]==incorr)]
    ax.scatter(subset["pca_x"], subset["pca_y"], subset["pca_z"], c=color,␣
  ↪s=20, alpha=0.8, label=f"{corp}, incorrect={incorr}")
ax.set_title("Average Embedding Value of Predictions, by Corpus", color="black")
ax.set_xlabel("PC1", color="black")
ax.set_ylabel("PC2", color="black")
ax.set_zlabel("PC3", color="black")
ax.legend(loc="best")
plt.show()
if "corpus" in df_test.columns:
    freqs = df_test["corpus"].value_counts()
    print("\nFrequency counts of corpus:", freqs)
    err_df = df_test[df_test["is_incorrect"]==True]
    corr_df = df_test[df_test["is_incorrect"]==False]
    err_counts = err_df["corpus"].value_counts()
    corr_counts = corr_df["corpus"].value_counts()
    print("\nCounts of corpus in misclassified:", err_counts)
    print("\nCounts of corpus in correctly classified:", corr_counts)
    print("\nProportions of corpus in misclassified:", err_counts/err_counts.
  ↪sum())
    print("\nProportions of corpus in correctly classified:", corr_counts/
  ↪corr_counts.sum())
    grouped_all = df_test.groupby("corpus")["avg_embedding"].apply(lambda x: np.
  ↪mean(np.stack(x.values), axis=0))
```

```
    grouped_err = err_df.groupby("corpus")["avg_embedding"].apply(lambda x: np.
↪mean(np.stack(x.values), axis=0))
    grouped_corr = corr_df.groupby("corpus")["avg_embedding"].apply(lambda x:␣
↪np.mean(np.stack(x.values), axis=0))
    # print("\nAvg embedding of each subcorpus overall:", grouped_all)
    # print("\nAvg embedding of each subcorpus misclassified:", grouped_err)
    # print("\nAvg embedding of each subcorpus correctly classified:",␣
↪grouped_corr)
err_stack = np.stack(df_test[df_test["is_incorrect"]==True]["avg_embedding"].
↪values)
corr_stack = np.stack(df_test[df_test["is_incorrect"]==False]["avg_embedding"].
↪values)
# print("\nAvg embedding of records predicted incorrectly:", err_stack.
↪mean(axis=0))
# print("Avg embedding of records predicted correctly:", corr_stack.
↪mean(axis=0))
error_df = df_test[df_test["is_incorrect"]==True].copy()
error_df.to_csv("misclassified_with_all_columns.csv", index=False)
error_save_path = os.path.join(dir_results,␣
↪f"{x_task}_{named_model}_{y_col}_{x_col}_errors.csv")
df_test.to_csv(error_save_path, index=False)
for corp in df_test["corpus"].unique():
    subset = df_test[df_test["corpus"]==corp]
    emb_true = subset[subset["is_incorrect"]==False]["avg_embedding"]
    emb_false = subset[subset["is_incorrect"]==True]["avg_embedding"]
    if len(emb_true)==0 or len(emb_false)==0:
        print(f"No valid data for subcorpus '{corp}'")
        continue
    p = np.mean(np.stack(emb_true.values), axis=0)
    q = np.mean(np.stack(emb_false.values), axis=0)
    p_exp = np.exp(p - np.max(p))
    q_exp = np.exp(q - np.max(q))
    p_sum = p_exp.sum()
    q_sum = q_exp.sum()
    if p_sum<=0 or q_sum<=0:
        print(f"Cannot form valid distributions for subcorpus '{corp}'")
        continue
    p_dist = p_exp / p_sum
    q_dist = q_exp / q_sum
    kl_pq = entropy(p_dist, q_dist)
    kl_qp = entropy(q_dist, p_dist)
    kl_sym = 0.5*(kl_pq + kl_qp)
    print(f"Subcorpus '{corp}' symmetric KL divergence: {kl_sym}")
error_save_path = os.path.join(dir_results,␣
↪f"{x_task}_{named_model}_{y_col}_{x_col}_errors.csv")
df_test.to_csv(error_save_path, index=False)
```

```
print("model results saved")
```

```
<IPython.core.display.HTML object>


Merged error_df with extra columns:

<IPython.core.display.HTML object>
```





```
Frequency counts of corpus: corpus
bible        88
europarl     87
biomed       75
Name: count, dtype: int64

Counts of corpus in misclassified: corpus
europarl     43
biomed       42
bible        40
Name: count, dtype: int64
```

```
Counts of corpus in correctly classified: corpus
bible       48
europarl    44
biomed      33
Name: count, dtype: int64

Proportions of corpus in misclassified: corpus
europarl    0.344
biomed      0.336
bible       0.320
Name: count, dtype: float64

Proportions of corpus in correctly classified: corpus
bible       0.384
europarl    0.352
biomed      0.264
Name: count, dtype: float64
Subcorpus 'bible' symmetric KL divergence: 1.888256876176576e-06
Subcorpus 'biomed' symmetric KL divergence: 1.6649774807960557e-06
Subcorpus 'europarl' symmetric KL divergence: 3.837849675757952e-06
model results saved
```

Result

0.2.5 answerdotai/ModernBERT-base

[ ]:
```
# Define Experiment Parameters
# named_model = "bert-base-cased"
# named_model = "roberta-base"
# named_model = "bert-large-cased"
# named_model = "roberta-large"
named_model = "answerdotai/ModernBERT-base" # modern bert
############
regularization_weight_decay = 0.5
learning_rate = 5e-6
size_batch = 128
length_max = 128
num_epochs = 1
###########################################
# x_col = "sentence"
x_col = "sentence_no_contractions"
# x_col = "pos_sequence"
# x_col = "dep_sequence"
# x_col = "morph_sequence"
# x_col = "snc_pos_seq"
# x_col = "snc_pos_alt"
# x_col = "snc_morph_seq"
# x_col = "snc_morph_alt"
# x_col = "snc_dep_seq"
# x_col = "snc_dep_alt"
# x_col = "snc_morph_complexity_value"
############
y_col = "binary_complexity"
# y_col = "complexity"
############
# x_task = "single"
x_task = "multi"
if x_task == "single":
    df_train = train_single_df
    df_val   = trial_val_single_df
    df_test  = test_single_df
else:
    df_train = train_multi_df
    df_val   = trial_val_multi_df
    df_test  = test_multi_df
##########################################
# Tokenize & Prepare Datasets
train_data_hf = prepare_dataset(
    df_train,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
val_data_hf = prepare_dataset(
    df_val,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
test_data_hf = prepare_dataset(
    df_test,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max)
print("Datasets prepared. Sample from train_data_hf:\n", train_data_hf[10])
# print("Datasets prepared. Sample from train_data_hf:\n", val_data_hf[10])
# print("Datasets prepared. Sample from train_data_hf:\n", test_data_hf[10])
##########################################
# custom_config = BertConfig.from_pretrained("roberta-base")
# custom_config.hidden_act = "gelu"  # alts: "relu" "silu"
# custom_config.attention_probs_dropout_prob = 0.1
# custom_config.hidden_dropout_prob = 0.1
# custom_config.gradient_checkpointing = False
```

```
##########################################
model, tokenizer = get_model_and_tokenizer(
    remote_model_name="answerdotai/ModernBERT-base",
    local_model_path=None,
    config=None)
############
# model, tokenizer = get_model_and_tokenizer(
#     remote_model_name=None
#     local_model_path="...CONFIGURE_PATH...",
#     config=custom_config)
print("=============")
print(named_model, ":")
print("=============")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters at load:", model.
  ↪num_parameters(only_trainable=True))
print("=============")
print("model lineage:", MODEL_LINEAGE)
print("=============")
```

```
Map:   0%|          | 0/1300 [00:00<?, ? examples/s]

Map:   0%|          | 0/250 [00:00<?, ? examples/s]

Map:   0%|          | 0/250 [00:00<?, ? examples/s]

Datasets prepared. Sample from train_data_hf:
 {'labels': tensor(0), 'input_ids': tensor([  101,  1109,  1210,  1808,  1104,
4836, 15949,   117,  1114,  1167,
         1190,   123,  1288,  8724,  1105, 24649,   117,  3657,  1107,  1103,
          163,  4626,  2758,  3293, 11171,   117,  1134, 11228, 11363,  1103,
         1352,  1104,  1103,  5655,  1206,  2470,  1386,  1105, 10585,   119,
          102,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0]),
'attention_mask': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0])}
Loading from Hugging Face model: answerdotai/ModernBERT-base
```

```
Some weights of ModernBertForSequenceClassification were not initialized from
the model checkpoint at answerdotai/ModernBERT-base and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

```
=============
answerdotai/ModernBERT-base :
=============
num_parameters: 149606402
num_trainable_parameters at load: 149606402
=============
model lineage: {'type': 'huggingface_hub', 'path': 'answerdotai/ModernBERT-
base', 'timestamp': '2025-04-14 01:54:41'}
=============
```

[ ]:
```
print(model)
```

```
ModernBertForSequenceClassification(
  (model): ModernBertModel(
    (embeddings): ModernBertEmbeddings(
      (tok_embeddings): Embedding(50368, 768, padding_idx=50283)
      (norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (drop): Dropout(p=0.0, inplace=False)
    )
    (layers): ModuleList(
      (0): ModernBertEncoderLayer(
        (attn_norm): Identity()
        (attn): ModernBertAttention(
          (Wqkv): Linear(in_features=768, out_features=2304, bias=False)
          (rotary_emb): ModernBertRotaryEmbedding()
          (Wo): Linear(in_features=768, out_features=768, bias=False)
          (out_drop): Identity()
        )
        (mlp_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (mlp): ModernBertMLP(
          (Wi): Linear(in_features=768, out_features=2304, bias=False)
          (act): GELUActivation()
          (drop): Dropout(p=0.0, inplace=False)
          (Wo): Linear(in_features=1152, out_features=768, bias=False)
        )
      )
      (1-21): 21 x ModernBertEncoderLayer(
        (attn_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (attn): ModernBertAttention(
          (Wqkv): Linear(in_features=768, out_features=2304, bias=False)
          (rotary_emb): ModernBertRotaryEmbedding()
          (Wo): Linear(in_features=768, out_features=768, bias=False)
          (out_drop): Identity()
        )
        (mlp_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (mlp): ModernBertMLP(
          (Wi): Linear(in_features=768, out_features=2304, bias=False)
          (act): GELUActivation()
          (drop): Dropout(p=0.0, inplace=False)
          (Wo): Linear(in_features=1152, out_features=768, bias=False)
        )
      )
    )
    (final_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  )
  (head): ModernBertPredictionHead(
    (dense): Linear(in_features=768, out_features=768, bias=False)
    (act): GELUActivation()
    (norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  )
  (drop): Dropout(p=0.0, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)
```

[ ]:
```
# for name, param in model.named_parameters():
#     print(name, "requires_grad=", param.requires_grad)
```

[ ]:
```
# # Inspect the attention_mask tensor for the first few samples
# for i in range(5):
#     print(train_data_hf[i]['attention_mask'])
```

[ ]:
```
##########################################
layers_to_unfreeze = [
    "model.layers.21.attn_norm.weight",
    "model.layers.21.attn.Wqkv.weight",
    "model.layers.21.attn.Wo.weight",
    "model.layers.21.mlp_norm.weight",
    "model.layers.21.mlp.Wi.weight",
    "model.layers.21.mlp.Wo.weight",
    "model.final_norm.weight",
    "head.dense.weight",
    "head.norm.weight",
    "classifier.weight",
    "classifier.bias"]
freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)
print(model.config)
print("=============")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
print("=============")
```

```
############################################
print("Experiment configuration used with this experiment:")
print("model used:", named_model)
print("learning rate used:", learning_rate)
print("number of epochs:", num_epochs)
print("maximum sequence length:", length_max)
print("batch size used:", size_batch)
print("regularization value:", regularization_weight_decay)
print("outcome variable:", y_col)
print("task:", x_task)
print("input column:", x_col)
############################################
print("=============")
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
```

```
ModernBertConfig {
  "_attn_implementation_autoset": true,
  "architectures": [
    "ModernBertForMaskedLM"
  ],
  "attention_bias": false,
  "attention_dropout": 0.0,
  "bos_token_id": 50281,
  "classifier_activation": "gelu",
  "classifier_bias": false,
  "classifier_dropout": 0.0,
  "classifier_pooling": "mean",
  "cls_token_id": 50281,
  "decoder_bias": true,
  "deterministic_flash_attn": false,
  "embedding_dropout": 0.0,
  "eos_token_id": 50282,
  "global_attn_every_n_layers": 3,
  "global_rope_theta": 160000.0,
  "gradient_checkpointing": false,
  "hidden_activation": "gelu",
  "hidden_size": 768,
  "initializer_cutoff_factor": 2.0,
  "initializer_range": 0.02,
  "intermediate_size": 1152,
  "layer_norm_eps": 1e-05,
  "local_attention": 128,
  "local_rope_theta": 10000.0,
  "max_position_embeddings": 8192,
  "mlp_bias": false,
  "mlp_dropout": 0.0,
  "model_type": "modernbert",
  "norm_bias": false,
  "norm_eps": 1e-05,
  "num_attention_heads": 12,
  "num_hidden_layers": 22,
  "pad_token_id": 50283,
  "position_embedding_type": "absolute",
  "reference_compile": null,
  "repad_logits_with_grad": false,
  "sep_token_id": 50282,
  "sparse_pred_ignore_index": -100,
  "sparse_prediction": false,
  "torch_dtype": "float32",
  "transformers_version": "4.50.3",
  "vocab_size": 50368
}

=============
num_parameters: 149606402
num_trainable_parameters: 5607938
=============
Experiment configuration used with this experiment:
model used: answerdotai/ModernBERT-base
learning rate used: 5e-06
number of epochs: 1
maximum sequence length: 128
batch size used: 128
regularization value: 0.5
outcome variable: binary_complexity
task: multi
input column: sentence_no_contractions
=============
num_trainable_parameters: 5607938
```

[ ]:
```
# for name, param in model.named_parameters():
#     print(name, "requires_grad=", param.requires_grad)
```

[ ]:
```
# model.resize_token_embeddings(len(tokenizer))
```

[ ]:
```
# Train & Evaluate
trained_model, trainer_obj = train_transformer_model(
    model = model,
    tokenizer = tokenizer,
    train_dataset = train_data_hf,
    val_dataset = val_data_hf,
    output_dir = dir_results,
    num_epochs = num_epochs,
    batch_size = size_batch,
    lr = learning_rate,
    weight_decay = regularization_weight_decay)
metrics = trainer_obj.evaluate()
print("Validation metrics:", metrics)
test_metrics = trainer_obj.evaluate(test_data_hf) if test_data_hf else None
print("Test metrics:", test_metrics)
```

```
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1611:
FutureWarning: `evaluation_strategy` is deprecated and will be removed in
version 4.46 of  Transformers. Use `eval_strategy` instead
  warnings.warn(
<ipython-input-20-81222a87c90b>:31: FutureWarning: `tokenizer` is deprecated and
will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class`
instead.
  trainer = Trainer(

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Validation metrics: {'eval_loss': 0.9965472221374512, 'eval_accuracy': 0.448,
'eval_precision': 0.42718446601941745, 'eval_recall': 0.8148148148148148,
'eval_f1': 0.5605095541401274, 'eval_runtime': 1.589, 'eval_samples_per_second':
157.33, 'eval_steps_per_second': 1.259, 'epoch': 1.0}
Test metrics: {'eval_loss': 0.9817145466804504, 'eval_accuracy': 0.504,
'eval_precision': 0.507537688442211, 'eval_recall': 0.7952755905511811,
'eval_f1': 0.6196319018404908, 'eval_runtime': 1.6323,
'eval_samples_per_second': 153.154, 'eval_steps_per_second': 1.225, 'epoch':
1.0}
```

[ ]:
```
# save model checkpoint
# timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
pacific_time = datetime.now(zoneinfo.ZoneInfo("America/Los_Angeles"))
timestamp = pacific_time.isoformat()
model_save_path = os.path.join(dir_models,␣
 ↪f"{x_task}_{named_model}_{y_col}_{timestamp}")
trainer_obj.save_model(model_save_path)
print(f"Model checkpoint saved to: {model_save_path}")
# log experiment results
experiment_info = {
    "model_name": named_model,
    "learning_rate": learning_rate,
    "epochs": num_epochs,
    "batch_size": size_batch,
    "weight_decay": regularization_weight_decay,
    "x_task": x_task,
    "x_col": x_col,
    "y_col": y_col,
    "layers_to_unfreeze": layers_to_unfreeze}
model_info = gather_model_details(trained_model)
```

```
all_run_metrics = gather_all_run_metrics(
    trainer=trainer_obj,
    train_dataset=train_data_hf,
    val_dataset=val_data_hf,
    test_dataset=test_data_hf)
log_experiment_results_json(
    experiment_meta=experiment_info,
    model_details=model_info,
    run_metrics=all_run_metrics,
    log_file=log_filepath)
print(f"EXPERIMENT LOGGED TO: {log_filepath}")
```

```
Model checkpoint saved to:
/content/drive/MyDrive/266-final/models/multi_answerdotai/ModernBERT-
base_binary_complexity_2025-04-13T18:54:57.678930-07:00
```

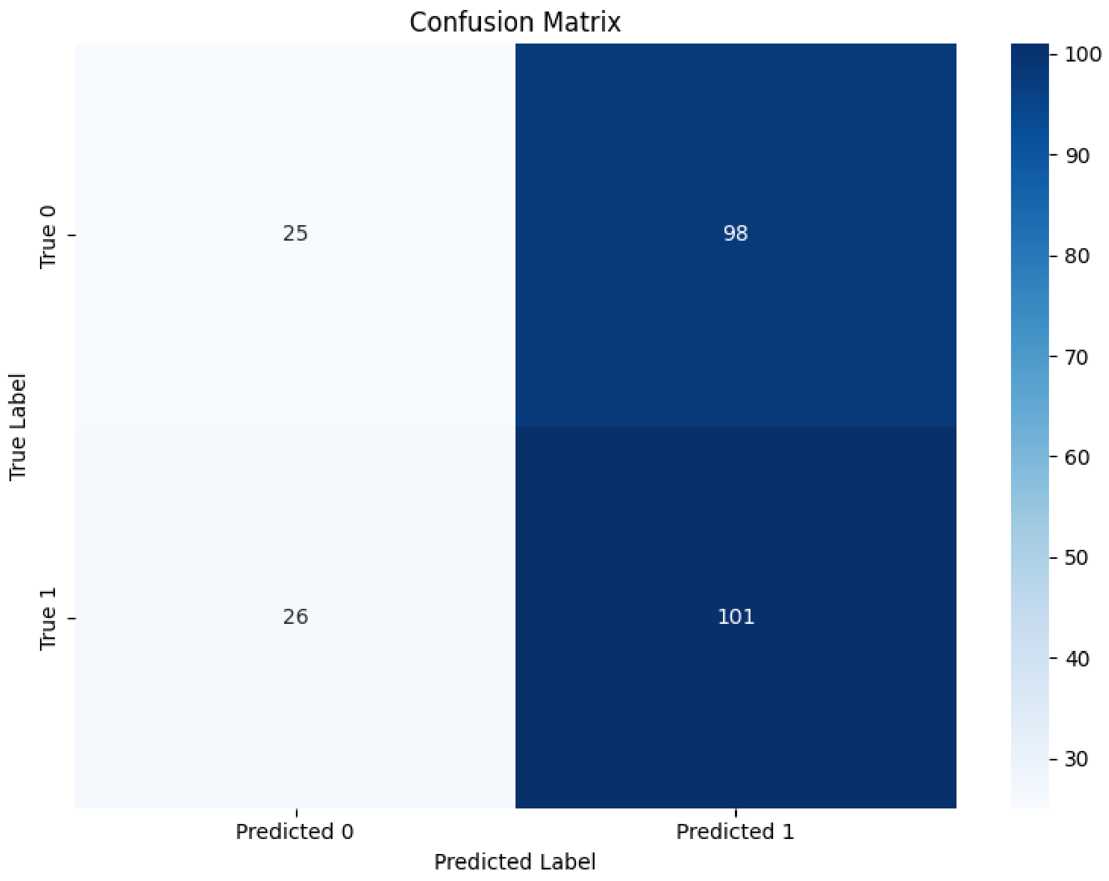
```
<IPython.core.display.HTML object>
```

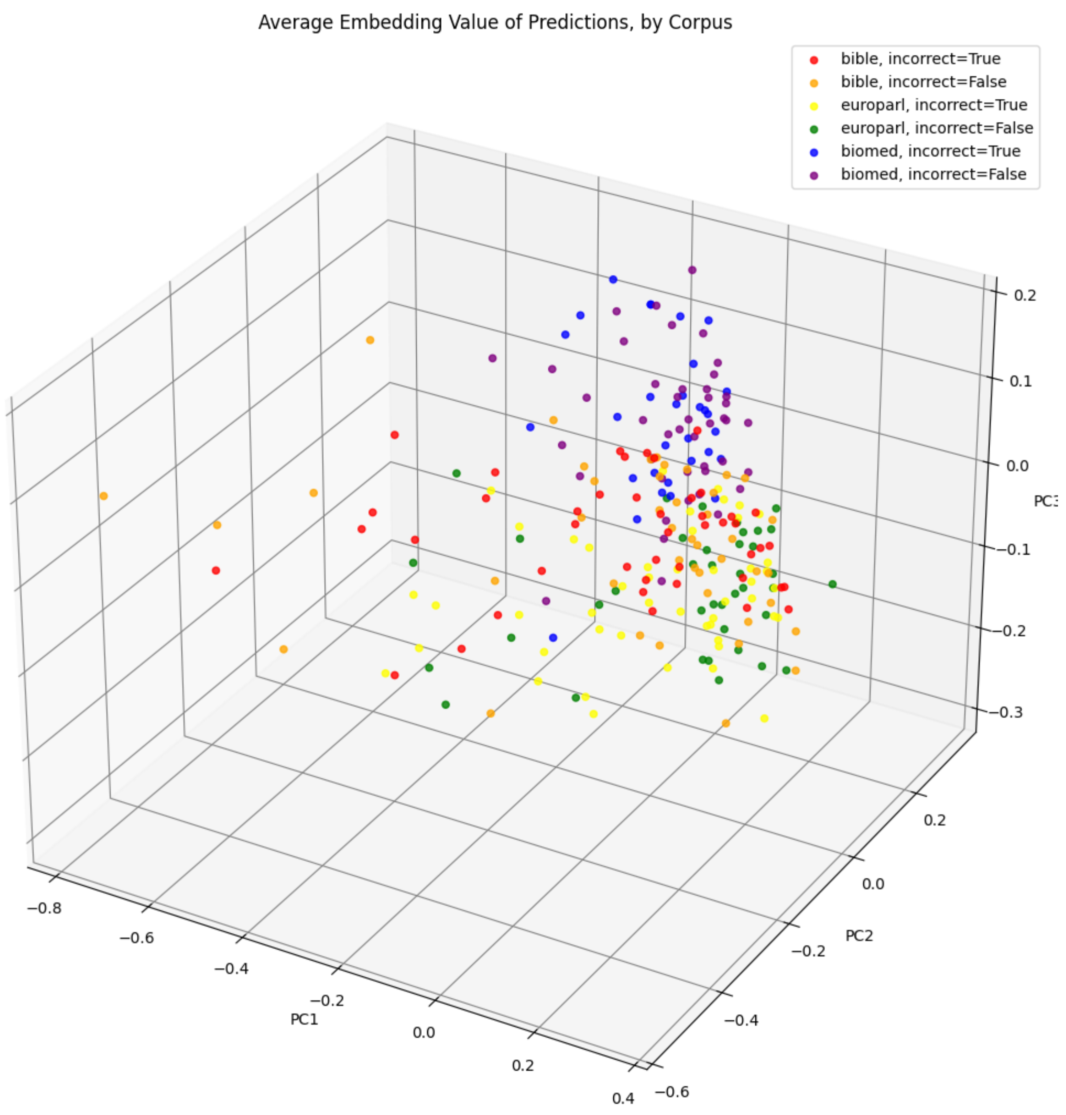
```
EXPERIMENT LOGGED TO:
/content/drive/MyDrive/266-final/results/experiment_runs.txt
```

[ ]:
```
prediction_output = trainer_obj.predict(test_data_hf)
raw_predictions = prediction_output.predictions
true_labels = prediction_output.label_ids
preds = np.argmax(raw_predictions, axis=1)
mismatch_indices = np.where(preds != true_labels)[0]

error_rows = []
for idx in mismatch_indices:
    text_value = df_test.iloc[idx][x_col]
    true_label_val = true_labels[idx]
    pred_label_val = preds[idx]

    error_rows.append({
        "hf_index": idx,
        "text": text_value,
        "true_label": true_label_val,
        "predicted_label": pred_label_val
    })

error_df = pd.DataFrame(error_rows)
df_test_for_merge = df_test.copy()
df_test_for_merge["error_matching_prefix"] = df_test_for_merge[x_col].str[:50]
# df_test_for_merge.drop(columns=[x_col], inplace=True)

error_df["error_matching_prefix"] = error_df["text"].str[:50]
error_df = error_df.merge(
    df_test_for_merge,
    on="error_matching_prefix",
    how="left",
    suffixes=("", "_source"))

error_df.to_csv("bert-base-cased_mismatches.csv", index=False)

# print("Number of misclassified samples:", len(error_df))
print("\nMerged error_df with extra columns:")
# display(error_df.head(15))

# print("\nConfusion Matrix:")
# cm = confusion_matrix(true_labels, preds)
# plt.figure(figsize=(8, 6))
# sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True,
#             xticklabels=["Predicted 0", "Predicted 1"],
#             yticklabels=["True 0", "True 1"])
# plt.xlabel('Predicted Label')
# plt.ylabel('True Label')
# plt.title('Confusion Matrix')
# plt.tight_layout()
# plt.show()
# print("confusion matrix metrics: \n", cm)
# error_save_path = os.path.join(dir_results,␣
 ↪f"{x_task}_{named_model}_{y_col}_{x_col}_errors.csv")
# error_df.to_csv(error_save_path, index=False)
# print("Result saved to results directory.")
prediction_output = trainer_obj.predict(test_data_hf)
raw_predictions = prediction_output.predictions
true_labels = prediction_output.label_ids
preds = np.argmax(raw_predictions, axis=1)
df_test["true_label"] = true_labels
df_test["predicted_label"] = preds
df_test["is_incorrect"] = (df_test["predicted_label"] != df_test["true_label"])
df_test["avg_embedding"] = None
device = next(model.parameters()).device
for i in range(len(df_test)):
    text_value = df_test.iloc[i][x_col]
    e = tokenizer(text_value, return_tensors="pt", truncation=True,␣
 ↪max_length=512).to(device)
    with torch.no_grad():
        # emb = model.bert.embeddings.word_embeddings(e["input_ids"]).
 ↪mean(dim=1).squeeze().cpu().numpy()
        emb = model.model.embeddings.tok_embeddings(e["input_ids"]).mean(dim=1).
 ↪squeeze().cpu().numpy()
    df_test.at[i,"avg_embedding"] = emb
cm = confusion_matrix(df_test["true_label"], df_test["predicted_label"])
plt.figure(figsize=(8,6))
```

```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True,␣
↪xticklabels=["Predicted 0","Predicted 1"], yticklabels=["True 0","True 1"])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.tight_layout()
plt.show()
# print("confusion matrix metrics:\n", cm)

df_plot = df_test.dropna(subset=["avg_embedding"]).copy()
embeddings = np.stack(df_plot["avg_embedding"].values)
pca = PCA(n_components=3)
reduced = pca.fit_transform(embeddings)
df_plot["pca_x"] = reduced[:,0]
df_plot["pca_y"] = reduced[:,1]
df_plot["pca_z"] = reduced[:,2]
colors = {
    ("bible", True): "red",
    ("bible", False): "orange",
    ("europarl", True): "yellow",
    ("europarl", False): "green",
    ("biomed", True): "blue",
    ("biomed", False): "purple"}
fig = plt.figure(figsize=(13,13))
ax = fig.add_subplot(111, projection='3d')
fig.set_facecolor("white")
ax.set_facecolor("white")
ax.xaxis._axinfo["grid"]["color"] = "gray"
ax.yaxis._axinfo["grid"]["color"] = "gray"
ax.zaxis._axinfo["grid"]["color"] = "gray"
for (corp, incorr), color in colors.items():
    subset = df_plot[(df_plot["corpus"]==corp) &␣
↪(df_plot["is_incorrect"]==incorr)]
    ax.scatter(subset["pca_x"], subset["pca_y"], subset["pca_z"], c=color,␣
↪s=20, alpha=0.8, label=f"{corp}, incorrect={incorr}")
ax.set_title("Average Embedding Value of Predictions, by Corpus", color="black")
ax.set_xlabel("PC1", color="black")
ax.set_ylabel("PC2", color="black")
ax.set_zlabel("PC3", color="black")
ax.legend(loc="best")
plt.show()
if "corpus" in df_test.columns:
    freqs = df_test["corpus"].value_counts()
    print("\nFrequency counts of corpus:", freqs)
    err_df = df_test[df_test["is_incorrect"]==True]
    corr_df = df_test[df_test["is_incorrect"]==False]
    err_counts = err_df["corpus"].value_counts()
    corr_counts = corr_df["corpus"].value_counts()
    print("\nCounts of corpus in misclassified:", err_counts)
    print("\nCounts of corpus in correctly classified:", corr_counts)
    print("\nProportions of corpus in misclassified:", err_counts/err_counts.
  ↪sum())
    print("\nProportions of corpus in correctly classified:", corr_counts/
  ↪corr_counts.sum())
    grouped_all = df_test.groupby("corpus")["avg_embedding"].apply(lambda x: np.
  ↪mean(np.stack(x.values), axis=0))
    grouped_err = err_df.groupby("corpus")["avg_embedding"].apply(lambda x: np.
  ↪mean(np.stack(x.values), axis=0))
    grouped_corr = corr_df.groupby("corpus")["avg_embedding"].apply(lambda x:␣
  ↪np.mean(np.stack(x.values), axis=0))
    # print("\nAvg embedding of each subcorpus overall:", grouped_all)
    # print("\nAvg embedding of each subcorpus misclassified:", grouped_err)
    # print("\nAvg embedding of each subcorpus correctly classified:",␣
  ↪grouped_corr)
err_stack = np.stack(df_test[df_test["is_incorrect"]==True]["avg_embedding"].
  ↪values)
corr_stack = np.stack(df_test[df_test["is_incorrect"]==False]["avg_embedding"].
  ↪values)
# print("\nAvg embedding of records predicted incorrectly:", err_stack.
  ↪mean(axis=0))
# print("Avg embedding of records predicted correctly:", corr_stack.
  ↪mean(axis=0))
error_df = df_test[df_test["is_incorrect"]==True].copy()
# error_df.to_csv("misclassified_with_all_columns.csv", index=False)
# error_save_path = os.path.join(dir_results,␣
  ↪f"{x_task}_modernbert-base_{y_col}_{x_col}_errors.csv")
df_test.to_csv(error_save_path, index=False)
for corp in df_test["corpus"].unique():
    subset = df_test[df_test["corpus"]==corp]
    emb_true = subset[subset["is_incorrect"]==False]["avg_embedding"]
    emb_false = subset[subset["is_incorrect"]==True]["avg_embedding"]
    if len(emb_true)==0 or len(emb_false)==0:
        print(f"No valid data for subcorpus '{corp}'")
        continue
    p = np.mean(np.stack(emb_true.values), axis=0)
    q = np.mean(np.stack(emb_false.values), axis=0)
    p_exp = np.exp(p - np.max(p))
    q_exp = np.exp(q - np.max(q))
    p_sum = p_exp.sum()
    q_sum = q_exp.sum()
    if p_sum<=0 or q_sum<=0:
        print(f"Cannot form valid distributions for subcorpus '{corp}'")
        continue
    p_dist = p_exp / p_sum
    q_dist = q_exp / q_sum
    kl_pq = entropy(p_dist, q_dist)
    kl_qp = entropy(q_dist, p_dist)
    kl_sym = 0.5*(kl_pq + kl_qp)
    print(f"Subcorpus '{corp}' symmetric KL divergence: {kl_sym}")
error_save_path = os.path.join(dir_results,␣
 ↪f"{x_task}_modernbert-base_{y_col}_{x_col}_errors.csv")
df_test.to_csv(error_save_path, index=False)
print("model results saved")
```

```
<IPython.core.display.HTML object>


Merged error_df with extra columns:

<IPython.core.display.HTML object>
```





```
Frequency counts of corpus: corpus
bible       88
europarl    87
biomed      75
Name: count, dtype: int64

Counts of corpus in misclassified: corpus
europarl    48
bible       45
biomed      31
Name: count, dtype: int64
```

```
Counts of corpus in correctly classified: corpus
biomed      44
bible       43
europarl    39
Name: count, dtype: int64

Proportions of corpus in misclassified: corpus
europarl    0.387097
bible       0.362903
biomed      0.250000
Name: count, dtype: float64

Proportions of corpus in correctly classified: corpus
biomed      0.349206
bible       0.341270
europarl    0.309524
Name: count, dtype: float64
Subcorpus 'bible' symmetric KL divergence: 8.12416918642706e-06
Subcorpus 'biomed' symmetric KL divergence: 6.612745022528127e-06
Subcorpus 'europarl' symmetric KL divergence: 1.2013919929424917e-05
model results saved
```

Result

0.2.6 answerdotai/ModernBERT-large

```
[ ]: # Define Experiment Parameters
     # named_model = "bert-base-cased"
     # named_model = "roberta-base"
     # named_model = "bert-large-cased"
     # named_model = "roberta-large"
     named_model = "answerdotai/ModernBERT-large" # modern bert
     ############
     regularization_weight_decay = 0.5
     learning_rate = 5e-6
     size_batch = 128
     length_max = 128
     num_epochs = 1
     ##########################################
     # x_col = "sentence"
     x_col = "sentence_no_contractions"
     # x_col = "pos_sequence"
     # x_col = "dep_sequence"
     # x_col = "morph_sequence"
     # x_col = "snc_pos_seq"
     # x_col = "snc_pos_alt"
     # x_col = "snc_morph_seq"
     # x_col = "snc_morph_alt"
     # x_col = "snc_dep_seq"
     # x_col = "snc_dep_alt"
     # x_col = "snc_morph_complexity_value"
     ############
     y_col = "binary_complexity"
     # y_col = "complexity"
     ############
     # x_task = "single"
     x_task = "multi"
     if x_task == "single":
         df_train = train_single_df
         df_val   = trial_val_single_df
         df_test  = test_single_df
     else:
         df_train = train_multi_df
         df_val   = trial_val_multi_df
         df_test  = test_multi_df
     ##########################################
     # Tokenize & Prepare Datasets
     train_data_hf = prepare_dataset(
         df_train,
         tokenizer,
         text_col=x_col,
         label_col=y_col,
         max_length=length_max)
     val_data_hf = prepare_dataset(
         df_val,
         tokenizer,
         text_col=x_col,
         label_col=y_col,
         max_length=length_max)
     test_data_hf = prepare_dataset(
         df_test,
         tokenizer,
         text_col=x_col,
         label_col=y_col,
         max_length=length_max)
     print("Datasets prepared. Sample from train_data_hf:\n", train_data_hf[10])
     # print("Datasets prepared. Sample from train_data_hf:\n", val_data_hf[10])
     # print("Datasets prepared. Sample from train_data_hf:\n", test_data_hf[10])
     ##########################################
     model, tokenizer = get_model_and_tokenizer(
         remote_model_name="answerdotai/ModernBERT-large",
         local_model_path=None,
         config=None)
     ############
```

```python
# model, tokenizer = get_model_and_tokenizer(
#     remote_model_name=None
#     local_model_path="...CONFIGURE_PATH...",
#     config=custom_config)
###########################################
print("=============")
print(named_model, ":")
print("=============")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters at load:", model.
 ↪num_parameters(only_trainable=True))
print("=============")
print("model lineage:", MODEL_LINEAGE)
print("=============")
```

```
Map:   0%|          | 0/1300 [00:00<?, ? examples/s]

Map:   0%|          | 0/250 [00:00<?, ? examples/s]

Map:   0%|          | 0/250 [00:00<?, ? examples/s]

Datasets prepared. Sample from train_data_hf:
 {'labels': tensor(0), 'input_ids': tensor([50281,   510,  1264,  2607,   273,
3638, 27155,    13,   342,   625,
          685,   374, 20181, 10071,   285, 36096,    84,    13,  7369,   275,
          253, 41379,  5477,  4019, 36729,    13,   534, 10534, 13806,   253,
         1375,   273,   253,  8881,   875,  8987,  5403,   285, 41460,    15,
        50282, 50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283,
        50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283,
        50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283,
        50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283,
        50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283,
        50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283,
        50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283,
        50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283,
        50283, 50283, 50283, 50283, 50283, 50283, 50283, 50283]),
'attention_mask': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0])}
Loading from Hugging Face model: answerdotai/ModernBERT-large

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

```
=============
answerdotai/ModernBERT-large :
=============
num_parameters: 395833346
num_trainable_parameters at load: 395833346
=============
model lineage: {'type': 'huggingface_hub', 'path': 'answerdotai/ModernBERT-
large', 'timestamp': '2025-04-14 01:55:15'}
=============
```

```
[ ]: print(model)
```

```
ModernBertForSequenceClassification(
  (model): ModernBertModel(
    (embeddings): ModernBertEmbeddings(
      (tok_embeddings): Embedding(50368, 1024, padding_idx=50283)
      (norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (drop): Dropout(p=0.0, inplace=False)
    )
    (layers): ModuleList(
      (0): ModernBertEncoderLayer(
        (attn_norm): Identity()
        (attn): ModernBertAttention(
          (Wqkv): Linear(in_features=1024, out_features=3072, bias=False)
          (rotary_emb): ModernBertRotaryEmbedding()
          (Wo): Linear(in_features=1024, out_features=1024, bias=False)
          (out_drop): Identity()
        )
        (mlp_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        (mlp): ModernBertMLP(
          (Wi): Linear(in_features=1024, out_features=5248, bias=False)
          (act): GELUActivation()
          (drop): Dropout(p=0.0, inplace=False)
          (Wo): Linear(in_features=2624, out_features=1024, bias=False)
        )
      )
      (1-27): 27 x ModernBertEncoderLayer(
        (attn_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        (attn): ModernBertAttention(
          (Wqkv): Linear(in_features=1024, out_features=3072, bias=False)
          (rotary_emb): ModernBertRotaryEmbedding()
          (Wo): Linear(in_features=1024, out_features=1024, bias=False)
          (out_drop): Identity()
        )
        (mlp_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        (mlp): ModernBertMLP(
          (Wi): Linear(in_features=1024, out_features=5248, bias=False)
          (act): GELUActivation()
          (drop): Dropout(p=0.0, inplace=False)
          (Wo): Linear(in_features=2624, out_features=1024, bias=False)
        )
      )
    )
    (final_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
  )
  (head): ModernBertPredictionHead(
    (dense): Linear(in_features=1024, out_features=1024, bias=False)
    (act): GELUActivation()
    (norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
  )
  (drop): Dropout(p=0.0, inplace=False)
  (classifier): Linear(in_features=1024, out_features=2, bias=True)
)
```

[ ]:
```
# for name, param in model.named_parameters():
#     print(name, "requires_grad=", param.requires_grad)
```

[ ]:
```
# # Inspect the attention_mask tensor for the first few samples
# for i in range(5):
#     print(train_data_hf[i]['attention_mask'])
```

[ ]:
```
##########################################
layers_to_unfreeze = [
    "model.layers.27.attn_norm.weight",
    "model.layers.27.attn.Wqkv.weight",
    "model.layers.27.attn.Wo.weight",
    "model.layers.27.mlp_norm.weight",
    "model.layers.27.mlp.Wi.weight",
    "model.layers.27.mlp.Wo.weight",
    "model.final_norm.weight",
    "head.dense.weight",
    "head.norm.weight",
    "classifier.weight",
    "classifier.bias"
    ]
freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)
print(model.config)
print("=============")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
print("=============")
##########################################
print("Experiment configuration used with this experiment:")
print("model used:", named_model)
print("learning rate used:", learning_rate)
```

```python
print("number of epochs:", num_epochs)
print("maximum sequence length:", length_max)
print("batch size used:", size_batch)
print("regularization value:", regularization_weight_decay)
print("outcome variable:", y_col)
print("task:", x_task)
print("input column:", x_col)
##########################################
print("=============")
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
```

```
ModernBertConfig {
  "_attn_implementation_autoset": true,
  "architectures": [
    "ModernBertForMaskedLM"
  ],
  "attention_bias": false,
  "attention_dropout": 0.0,
  "bos_token_id": 50281,
  "classifier_activation": "gelu",
  "classifier_bias": false,
  "classifier_dropout": 0.0,
  "classifier_pooling": "mean",
  "cls_token_id": 50281,
  "decoder_bias": true,
  "deterministic_flash_attn": false,
  "embedding_dropout": 0.0,
  "eos_token_id": 50282,
  "global_attn_every_n_layers": 3,
  "global_rope_theta": 160000.0,
  "gradient_checkpointing": false,
  "hidden_activation": "gelu",
  "hidden_size": 1024,
  "initializer_cutoff_factor": 2.0,
  "initializer_range": 0.02,
  "intermediate_size": 2624,
  "layer_norm_eps": 1e-05,
  "local_attention": 128,
  "local_rope_theta": 10000.0,
  "max_position_embeddings": 8192,
  "mlp_bias": false,
  "mlp_dropout": 0.0,
  "model_type": "modernbert",
  "norm_bias": false,
  "norm_eps": 1e-05,
  "num_attention_heads": 16,
  "num_hidden_layers": 28,
  "pad_token_id": 50283,
  "position_embedding_type": "absolute",
  "reference_compile": null,
  "repad_logits_with_grad": false,
  "sep_token_id": 50282,
  "sparse_pred_ignore_index": -100,
  "sparse_prediction": false,
  "torch_dtype": "float32",
  "transformers_version": "4.50.3",
  "vocab_size": 50368
}

=============
num_parameters: 395833346
num_trainable_parameters: 13309954
=============
Experiment configuration used with this experiment:
model used: answerdotai/ModernBERT-large
learning rate used: 5e-06
number of epochs: 1
maximum sequence length: 128
batch size used: 128
regularization value: 0.5
outcome variable: binary_complexity
task: multi
input column: sentence_no_contractions
=============
num_trainable_parameters: 13309954
```

[ ]:
```
# for name, param in model.named_parameters():
#     print(name, "requires_grad=", param.requires_grad)
```

[ ]:
```
# model.resize_token_embeddings(len(tokenizer))
```

[ ]:
```
# Train & Evaluate
trained_model, trainer_obj = train_transformer_model(
    model = model,
    tokenizer = tokenizer,
    train_dataset = train_data_hf,
    val_dataset = val_data_hf,
    output_dir = dir_results,
    num_epochs = num_epochs,
    batch_size = size_batch,
    lr = learning_rate,
    weight_decay = regularization_weight_decay)
metrics = trainer_obj.evaluate()
print("Validation metrics:", metrics)
test_metrics = trainer_obj.evaluate(test_data_hf) if test_data_hf else None
```

```
print("Test metrics:", test_metrics)
```

```
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1611:
FutureWarning: `evaluation_strategy` is deprecated and will be removed in
version 4.46 of  Transformers. Use `eval_strategy` instead
  warnings.warn(
<ipython-input-20-81222a87c90b>:31: FutureWarning: `tokenizer` is deprecated and
will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class`
instead.
  trainer = Trainer(
```

```
<IPython.core.display.HTML object>
```

```
W0414 01:55:25.351000 17035 torch/_dynamo/convert_frame.py:906] [1/8]
torch._dynamo hit config.cache_size_limit (8)
W0414 01:55:25.351000 17035 torch/_dynamo/convert_frame.py:906] [1/8]
function: 'compiled_mlp' (/usr/local/lib/python3.11/dist-
packages/transformers/models/modernbert/modeling_modernbert.py:552)
W0414 01:55:25.351000 17035 torch/_dynamo/convert_frame.py:906] [1/8]    last
reason: 1/0: GLOBAL_STATE changed: grad_mode
W0414 01:55:25.351000 17035 torch/_dynamo/convert_frame.py:906] [1/8] To log all
recompilation reasons, use TORCH_LOGS="recompiles".
W0414 01:55:25.351000 17035 torch/_dynamo/convert_frame.py:906] [1/8] To
diagnose recompilation issues, see
https://pytorch.org/docs/main/torch.compiler_troubleshooting.html.
```

```
<IPython.core.display.HTML object>
```

```
Validation metrics: {'eval_loss': 0.7263299822807312, 'eval_accuracy': 0.524,
'eval_precision': 0.4678362573099415, 'eval_recall': 0.7407407407407407,
'eval_f1': 0.5734767025089605, 'eval_runtime': 2.4012,
'eval_samples_per_second': 104.115, 'eval_steps_per_second': 0.833, 'epoch':
1.0}
Test metrics: {'eval_loss': 0.7307943105697632, 'eval_accuracy': 0.54,
'eval_precision': 0.5357142857142857, 'eval_recall': 0.7086614173228346,
'eval_f1': 0.6101694915254238, 'eval_runtime': 12.8892,
'eval_samples_per_second': 19.396, 'eval_steps_per_second': 0.155, 'epoch': 1.0}
```

[ ]:
```
# save model checkpoint
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
model_save_path = os.path.join(dir_models,␣
 ↪f"{x_task}_{named_model}_{y_col}_{timestamp}")
trainer_obj.save_model(model_save_path)
print(f"Model checkpoint saved to: {model_save_path}")
# log experiment results
experiment_info = {
    "model_name": named_model,
    "learning_rate": learning_rate,
    "epochs": num_epochs,
    "batch_size": size_batch,
    "weight_decay": regularization_weight_decay,
    "x_task": x_task,
    "x_col": x_col,
    "y_col": y_col,
    "layers_to_unfreeze": layers_to_unfreeze}
model_info = gather_model_details(trained_model)
all_run_metrics = gather_all_run_metrics(
    trainer=trainer_obj,
    train_dataset=train_data_hf,
    val_dataset=val_data_hf,
    test_dataset=test_data_hf)
log_experiment_results_json(
    experiment_meta=experiment_info,
    model_details=model_info,
    run_metrics=all_run_metrics,
    log_file=log_filepath)
print(f"EXPERIMENT LOGGED TO: {log_filepath}")
```

```
Model checkpoint saved to:
/content/drive/MyDrive/266-final/models/multi_answerdotai/ModernBERT-
large_binary_complexity_20250414_015543
```

```
<IPython.core.display.HTML object>
```

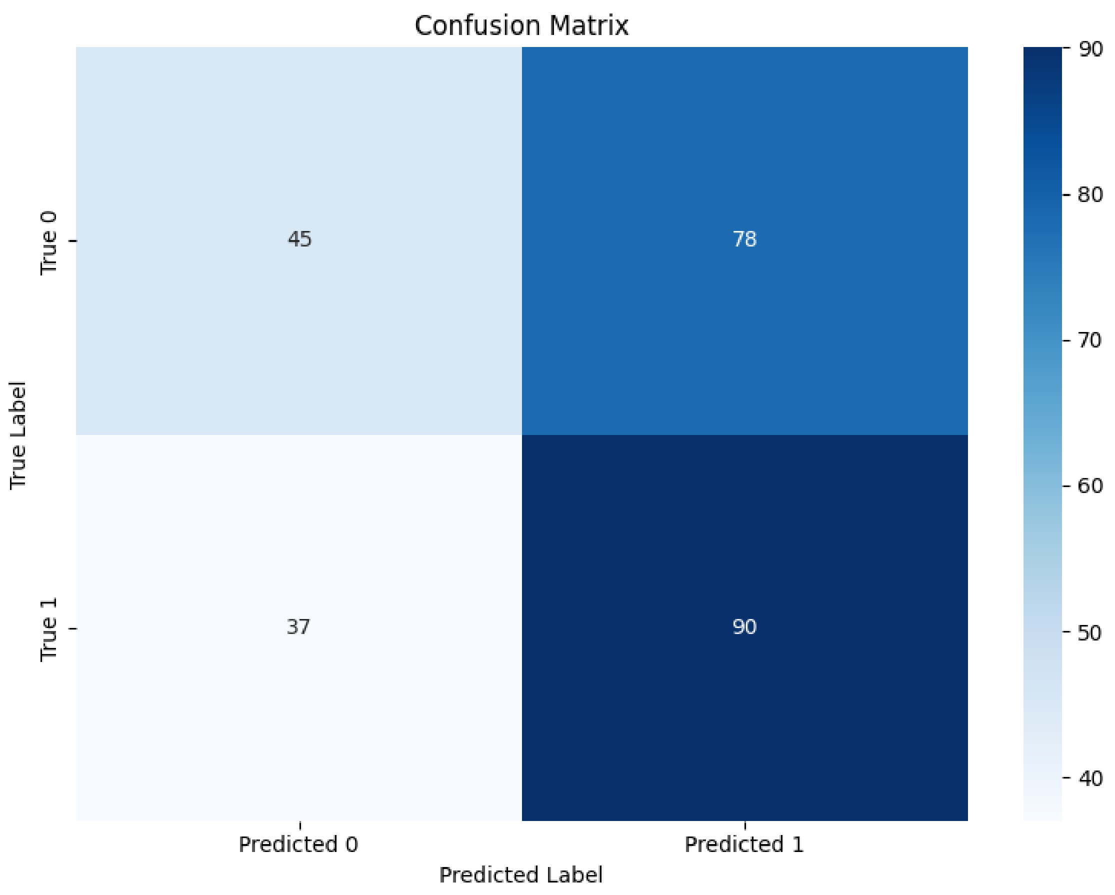
```
EXPERIMENT LOGGED TO:
/content/drive/MyDrive/266-final/results/experiment_runs.txt
```

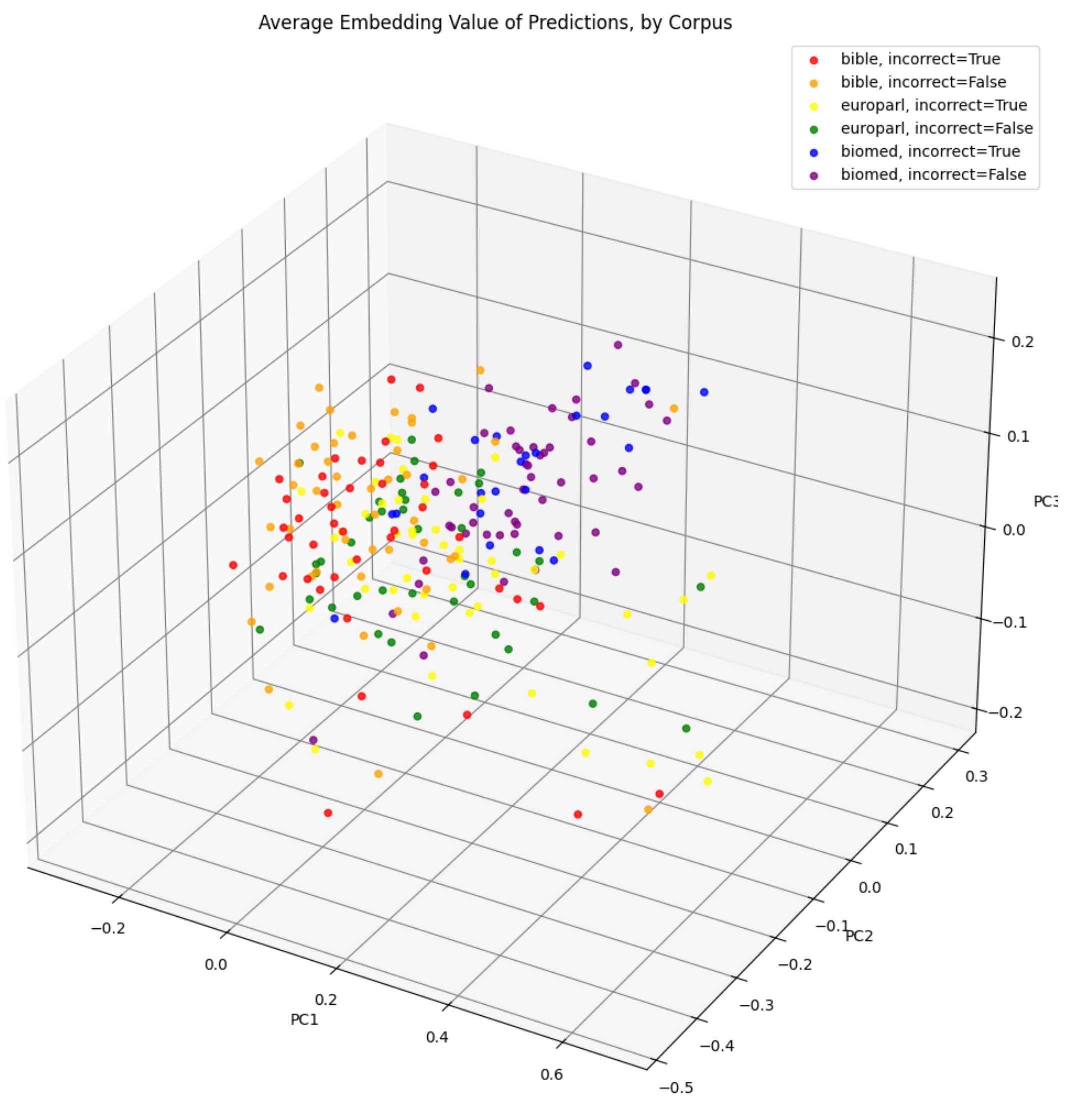
[ ]:
```
prediction_output = trainer_obj.predict(test_data_hf)
raw_predictions = prediction_output.predictions
true_labels = prediction_output.label_ids
preds = np.argmax(raw_predictions, axis=1)
mismatch_indices = np.where(preds != true_labels)[0]

error_rows = []
for idx in mismatch_indices:
    text_value = df_test.iloc[idx][x_col]
    true_label_val = true_labels[idx]
    pred_label_val = preds[idx]

    error_rows.append({
        "hf_index": idx,
        "text": text_value,
        "true_label": true_label_val,
        "predicted_label": pred_label_val
    })

error_df = pd.DataFrame(error_rows)
```

```
df_test_for_merge = df_test.copy()
df_test_for_merge["error_matching_prefix"] = df_test_for_merge[x_col].str[:50]
# df_test_for_merge.drop(columns=[x_col], inplace=True)

error_df["error_matching_prefix"] = error_df["text"].str[:50]
error_df = error_df.merge(
    df_test_for_merge,
    on="error_matching_prefix",
    how="left",
    suffixes=("", "_source"))

error_df.to_csv("bert-base-cased_mismatches.csv", index=False)

# print("Number of misclassified samples:", len(error_df))
print("\nMerged error_df with extra columns:")
# display(error_df.head(15))

# print("\nConfusion Matrix:")
# cm = confusion_matrix(true_labels, preds)
# plt.figure(figsize=(8, 6))
# sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True,
#             xticklabels=["Predicted 0", "Predicted 1"],
#             yticklabels=["True 0", "True 1"])
# plt.xlabel('Predicted Label')
# plt.ylabel('True Label')
# plt.title('Confusion Matrix')
# plt.tight_layout()
# plt.show()
# print("confusion matrix metrics: \n", cm)
# error_save_path = os.path.join(dir_results,␣
 ↪f"{x_task}_{named_model}_{y_col}_{x_col}_errors.csv")
# error_df.to_csv(error_save_path, index=False)
# print("Result saved to results directory.")
prediction_output = trainer_obj.predict(test_data_hf)
raw_predictions = prediction_output.predictions
true_labels = prediction_output.label_ids
preds = np.argmax(raw_predictions, axis=1)
df_test["true_label"] = true_labels
df_test["predicted_label"] = preds
df_test["is_incorrect"] = (df_test["predicted_label"] != df_test["true_label"])
df_test["avg_embedding"] = None
device = next(model.parameters()).device
for i in range(len(df_test)):
    text_value = df_test.iloc[i][x_col]
    e = tokenizer(text_value, return_tensors="pt", truncation=True,␣
 ↪max_length=512).to(device)
    with torch.no_grad():
        # emb = model.bert.embeddings.word_embeddings(e["input_ids"]).
↪mean(dim=1).squeeze().cpu().numpy()
        emb = model.model.embeddings.tok_embeddings(e["input_ids"]).mean(dim=1).
↪squeeze().cpu().numpy()
    df_test.at[i,"avg_embedding"] = emb
cm = confusion_matrix(df_test["true_label"], df_test["predicted_label"])
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True,␣
↪xticklabels=["Predicted 0","Predicted 1"], yticklabels=["True 0","True 1"])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.tight_layout()
plt.show()
# print("confusion matrix metrics:\n", cm)

df_plot = df_test.dropna(subset=["avg_embedding"]).copy()
embeddings = np.stack(df_plot["avg_embedding"].values)
pca = PCA(n_components=3)
reduced = pca.fit_transform(embeddings)
df_plot["pca_x"] = reduced[:,0]
df_plot["pca_y"] = reduced[:,1]
df_plot["pca_z"] = reduced[:,2]
colors = {
    ("bible", True): "red",
    ("bible", False): "orange",
    ("europarl", True): "yellow",
    ("europarl", False): "green",
    ("biomed", True): "blue",
    ("biomed", False): "purple"}
fig = plt.figure(figsize=(13,13))
ax = fig.add_subplot(111, projection='3d')
fig.set_facecolor("white")
ax.set_facecolor("white")
ax.xaxis._axinfo["grid"]["color"] = "gray"
ax.yaxis._axinfo["grid"]["color"] = "gray"
ax.zaxis._axinfo["grid"]["color"] = "gray"
for (corp, incorr), color in colors.items():
    subset = df_plot[(df_plot["corpus"]==corp) &␣
↪(df_plot["is_incorrect"]==incorr)]
    ax.scatter(subset["pca_x"], subset["pca_y"], subset["pca_z"], c=color,␣
↪s=20, alpha=0.8, label=f"{corp}, incorrect={incorr}")
ax.set_title("Average Embedding Value of Predictions, by Corpus", color="black")
ax.set_xlabel("PC1", color="black")
ax.set_ylabel("PC2", color="black")
ax.set_zlabel("PC3", color="black")
ax.legend(loc="best")
```

```
plt.show()
if "corpus" in df_test.columns:
    freqs = df_test["corpus"].value_counts()
    print("\nFrequency counts of corpus:", freqs)
    err_df = df_test[df_test["is_incorrect"]==True]
    corr_df = df_test[df_test["is_incorrect"]==False]
    err_counts = err_df["corpus"].value_counts()
    corr_counts = corr_df["corpus"].value_counts()
    print("\nCounts of corpus in misclassified:", err_counts)
    print("\nCounts of corpus in correctly classified:", corr_counts)
    print("\nProportions of corpus in misclassified:", err_counts/err_counts.
 ↪sum())
    print("\nProportions of corpus in correctly classified:", corr_counts/
 ↪corr_counts.sum())
    grouped_all = df_test.groupby("corpus")["avg_embedding"].apply(lambda x: np.
 ↪mean(np.stack(x.values), axis=0))
    grouped_err = err_df.groupby("corpus")["avg_embedding"].apply(lambda x: np.
 ↪mean(np.stack(x.values), axis=0))
    grouped_corr = corr_df.groupby("corpus")["avg_embedding"].apply(lambda x:␣
 ↪np.mean(np.stack(x.values), axis=0))
    # print("\nAvg embedding of each subcorpus overall:", grouped_all)
    # print("\nAvg embedding of each subcorpus misclassified:", grouped_err)
    # print("\nAvg embedding of each subcorpus correctly classified:",␣
 ↪grouped_corr)
err_stack = np.stack(df_test[df_test["is_incorrect"]==True]["avg_embedding"].
 ↪values)
corr_stack = np.stack(df_test[df_test["is_incorrect"]==False]["avg_embedding"].
 ↪values)
# print("\nAvg embedding of records predicted incorrectly:", err_stack.
 ↪mean(axis=0))
# print("Avg embedding of records predicted correctly:", corr_stack.
 ↪mean(axis=0))
error_df = df_test[df_test["is_incorrect"]==True].copy()
# error_df.to_csv("misclassified_with_all_columns.csv", index=False)
# error_save_path = os.path.join(dir_results,␣
 ↪f"{x_task}_modernbert-base_{y_col}_{x_col}_errors.csv")
df_test.to_csv(error_save_path, index=False)
for corp in df_test["corpus"].unique():
    subset = df_test[df_test["corpus"]==corp]
    emb_true = subset[subset["is_incorrect"]==False]["avg_embedding"]
    emb_false = subset[subset["is_incorrect"]==True]["avg_embedding"]
    if len(emb_true)==0 or len(emb_false)==0:
        print(f"No valid data for subcorpus '{corp}'")
        continue
    p = np.mean(np.stack(emb_true.values), axis=0)
    q = np.mean(np.stack(emb_false.values), axis=0)
    p_exp = np.exp(p - np.max(p))
    q_exp = np.exp(q - np.max(q))
    p_sum = p_exp.sum()
    q_sum = q_exp.sum()
    if p_sum<=0 or q_sum<=0:
        print(f"Cannot form valid distributions for subcorpus '{corp}'")
        continue
    p_dist = p_exp / p_sum
    q_dist = q_exp / q_sum
    kl_pq = entropy(p_dist, q_dist)
    kl_qp = entropy(q_dist, p_dist)
    kl_sym = 0.5*(kl_pq + kl_qp)
    print(f"Subcorpus '{corp}' symmetric KL divergence: {kl_sym}")
error_save_path = os.path.join(dir_results,␣
 ↪f"{x_task}_modernbert-large_{y_col}_{x_col}_errors.csv")
df_test.to_csv(error_save_path, index=False)
print("model results saved")
```

```
<IPython.core.display.HTML object>


Merged error_df with extra columns:

<IPython.core.display.HTML object>
```


Confusion Matrix
True 0
True 1
True Label
45
78
37
90
Predicted 0
Predicted 1
Predicted Label
90
80
70
60
50
40



```
Frequency counts of corpus: corpus
bible        88
europarl     87
biomed       75
Name: count, dtype: int64

Counts of corpus in misclassified: corpus
europarl     47
bible        42
biomed       26
Name: count, dtype: int64
```

```
Counts of corpus in correctly classified: corpus
biomed      49
bible       46
europarl    40
Name: count, dtype: int64

Proportions of corpus in misclassified: corpus
europarl    0.408696
bible       0.365217
biomed      0.226087
Name: count, dtype: float64

Proportions of corpus in correctly classified: corpus
biomed      0.362963
bible       0.340741
europarl    0.296296
Name: count, dtype: float64
Subcorpus 'bible' symmetric KL divergence: 4.29344419874927e-06
Subcorpus 'biomed' symmetric KL divergence: 3.5779938949494955e-06
Subcorpus 'europarl' symmetric KL divergence: 4.058646832767972e-06
model results saved
```

Result

```
[ ]:
```